Facultad
de
Ciencias

DISEÑO E IMPLEMENTACIÓN DE UN MOTOR DE MACHINE LEARNING MULTIPLATAFORMA Y SU INTEGRACIÓN EN UN PRODUCTO PARA LA INTELIGENCIA OPERACIONAL

(Design and implementation of a multiplatform Machine Learning engine and its inclusion in an operational intelligence product)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Fernando Solar Iglesias

Director: Patricia López Martínez

Codirector: David Marín Oujo

Septiembre - 2019

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Con la presentación de este trabajo, y por tanto acceso al Grado en Ingeniería Informática, me gustaría **agradecer con estas líneas a todas aquellas personas que han formado parte durante estos años de esta etapa**.

Muy **especialmente a mi familia y amigos**, por el incalculable valor que supone su confianza y apoyo desinteresado. Estoy seguro de que sin ellos hubiese sido imposible alcanzar todos los objetivos.

A **CIC Consulting Informático y al equipo de IDbox** por permitirme desarrollar este proyecto en la empresa e integrarlo en su consolidado producto. En concreto, agradecer a David Marín su impecable labor como codirector del proyecto; es realmente un privilegio y supone un gran aprendizaje el poder trabajar a su lado. Igualmente a Marta García por su disposición y ayuda en IDbox; creo que sus ganas por el buen hacer y su energía resultan muy positivas para todos.

También a los **docentes de la Universidad de Cantabria** que han dado sentido a los distintas asignaturas del grado. En particular en esta tarea, a Patricia López por su compromiso y calidad como directora del trabajo, cuyas aportaciones realizadas al proyecto han resultado de gran valor.

Por último y no menos importante, dar las gracias al **equipo que me ha acompañado en mis proyectos *software* personales** realizados en paralelo a los del grado y la empresa. Sin ellos no hubiera sido posible llevar a cabo las ideas que me permitieron adquirir destrezas utilizadas en este proyecto.

A todos, **gracias por vuestra aportación** en este ámbito que para mí significa más que un trabajo o unos estudios.

Resumen

Gracias a la gran cantidad de librerías disponibles y a la activa comunidad de usuarios y desarrolladores, **Python se ha convertido en una de las plataformas preferidas para la implementación de Machine Learning**. No obstante, **a la hora de elaborar sistemas de gestión en la empresa se encuentran asentados otros entornos como .NET de Microsoft**. Esta tecnología es la usada por el producto **IDbox de CIC Consulting Informático** para permitir la **supervisión y análisis en tiempo real o históricos de procesos de negocio** en distintos ámbitos: industria, energía y entornos inteligentes.

En este contexto de inteligencia operacional, se consideró interesante la posibilidad de **introducir aprendizaje automático desarrollado en Python como una de las operaciones de procesamiento de los datos**. Para afrontar esta interoperatibilidad entre tecnologías, sin que afecte al funcionamiento del caso de uso por parte del usuario, **se presenta como solución el diseño a medida e implementación de un motor de Machine Learning y su integración con IDbox**. Para ello se abordan una serie de trabajos, como el análisis del problema y las tecnologías involucradas, la intercomunicación de procesos a nivel de sistema operativo, el ciclo de vida completo de las operaciones y datos, así como los canales de transmisión adecuados para la comunicación de grandes cantidades de datos estructurados.

Palabras clave: Motor multiplataforma, aprendizaje automático, inteligencia operacional, C#, Python.

Abstract

Due to the large number of available libraries and its strong community of users and developers, **Python has become one of the most in-demand platforms for Machine Learning implementation**. However, **when it comes to develop management systems in a company, other environments such as Microsoft .NET** are established. This technology is used by **IDbox - a product created by CIC Consulting Informático -** to allow **real-time or historical monitoring and analysis of business processes** in different sectors: industry, energy and smart.

The fact of **introducing Machine Learning algorithms developed in Python as one of the data processing operations** was considered interesting in this operational intelligence context. To address the interoperability problem between technologies, without affecting the user case workflow, **a custom design and implementation of a Machine Learning engine and its integration with IDbox** is introduced. In order to achieve this goal, a series of steps are followed, such as the analysis of the problem and the technologies involved, the intercommunication of processes at the operating system level, the complete life cycle of operations and data, as well as the appropriate transmission channel to communicate large amounts of structured data.

Keywords: Cross-platform engine, machine learning, operational intelligence, C#, Python.

Capítulo 1

Introducción

Debido a los avances tecnológicos que se vienen produciendo en la última década, la sociedad se encuentra inmersa en una **transformación digital** que se hace notar en sus distintos ámbitos: entidades, gobiernos, empresas e individuos han visto su forma de trabajar adaptada a la nueva generación digital.

Tanto es así que, ahora más que nunca, **los datos se han convertido en uno de los aspectos de mayor importancia de la sociedad**. Sin darnos cuenta en la mayoría de las ocasiones, un porcentaje significativo de las acciones que realizamos en nuestro día a día contribuyen al crecimiento de este volumen total de datos. Aunque no es posible conocer dicha cifra con exactitud, debido a la gran fragmentación de sus orígenes y de las plataformas sobre las que se almacenan, distintos estudios y análisis coinciden en dos aspectos clave: tan solo la cantidad de datos generados durante estos últimos años supera ya a la producida desde comienzos de siglo, y el orden de magnitud utilizado en el presente se sitúa en el *zettabyte*.

Prefijo	**Nomenclatura**	**Representación decimal**	**Año de asignación**
Mega	Millón	1.000.000	1960
Giga	Billón	1.000.000.000	1960
Tera	Trillón	1.000.000.000.000	1960
Peta	Cuatrillón	1.000.000.000.000.000	1975
Exa	Quintillón	1.000.000.000.000.000.000	1975
Zetta	**Sextillón**	**1.000.000.000.000.000.000.000**	**1991**
Yotta	Septillón	1.000.000.000.000.000.000.000.000	1991

Cuadro 1.1: Órdenes de magnitud reconocidos por el Sistema Internacional

Como muestra el cuadro 1.1, desde su última actualización en 1991, tan solo un nivel de orden de magnitud del Sistema Internacional quedaría disponible para describir la abundante suma de datos producida, y es que los términos *peta* y *exa* se han quedado atrás en cuanto a ser una opción apropiada para especificar la situación actual [1].

De cara al futuro y en coherencia con este rápido crecimiento, **la previsión sobre la magnitud total próxima es superior**. El constante incremento de dispositivos con capacidad para comunicarse y la expansión de la conectividad a internet, tanto por nuevos accesos como por la mejora de la calidad de la red actual, son sin duda dos de los factores principales que evidencian dicho porvenir.

En paralelo a esta voluminosa generación de datos, debates sobre cómo, por quién, y dónde manipularlos están a la orden del día. Sin embargo, existe un punto de encuentro compartido: un **uso autorizado y responsable de los datos** es imprescindible para **convertirlos en información valiosa** que pueda aportar nuevas conclusiones y toma de decisiones justificadas.

"If you can not measure it, you can not improve it."
William Thomson, Lord Kelvin

1.1. Integración, procesamiento y análisis de datos desde IDbox

Para ayudar a convertir un gran volumen de datos en información útil surgió en el mercado **IDbox RT**, una aplicación informática cuyos orígenes se remontan a comienzos del año 2000 en **CIC Consulting Informático**: una empresa de ingeniería y desarrollo de proyectos en el contexto de la informática y las comunicaciones con cerca de 30 años de experiencia en el sector.

IDbox permite la supervisión y el análisis en tiempo real o históricos de distintos procesos de negocio en diferentes ámbitos y sectores como las líneas de producción y eficiencia energética de la **industria**, la generación y distribución de **energía**, o los **entornos inteligentes** como las *smartcities* y redes de suministro, entre otros [2]. Para ello, sus componentes *software* modulares cooperan en base al siguiente flujo de trabajo:

1. **Integración**: en este punto inicial tiene lugar la recogida de datos de diferentes orígenes físicos o virtuales, conectándose a ellos para recibir su flujo de datos. Ejemplos de artefactos habituales en esta etapa son bases de datos, PLCs, sensores, servicios web, ficheros, sistemas, entre otros.

2. **Procesamiento**: posteriormente, y en base a las reglas de negocio definidas, se realiza el procesamiento de los datos recogidos, sobre los que tiene lugar la realización de cálculos y el empleo de técnicas de *big data* para producir nuevos valores e información útiles.

3. **Análisis**: con los datos recogidos y procesados, el usuario final dispone de distintas herramientas de visualización y análisis para facilitar la toma de decisiones desde dispositivos de escritorio o móviles. Sinópticos, gráficas, mapas, estadísticas, cuadros de mando, alarmas, informes o notificaciones son algunos de los elementos que forman parte de esta centralización de la información.

A través de estos pasos [3], la plataforma permite dotar a una organización de las herramientas necesarias para conocer qué está ocurriendo con sus procesos de negocio y optimizarlos, logrando como consecuencia el aumento de la productividad y la aportación de valor al flujo de trabajo, esto es: la **inteligencia operacional** [4].

1.2. La importancia del Machine Learning en la generación de información

El aprendizaje automático, traducción del término en inglés *Machine Learning*, se enmarca dentro de la **inteligencia artificial** y forma parte de las distintas operaciones de *big data* que se aplican en la fase de procesamiento de datos anteriormente descrita.

Son múltiples las aplicaciones que pueden obtenerse mediante su uso, siendo la **solución a varios problemas populares** como la clasificación (dado un elemento discernir cuál es su categoría), regresión (obtención de un valor respecto a un conjunto de datos de entrada), asociación (identificación de patrones sobre relaciones entre datos) o agrupamiento (organización de un conjunto de datos por su naturaleza), entre otros.

En el contexto descrito en este trabajo, resulta interesante la obtención de nueva información que permite la **regresión lineal en base a valores continuos** previamente conocidos. Estos datos se disponen ordenados en una colección con forma de matriz, popularmente denominada *dataset*, donde cada columna de esta representación matemática define un tipo distinto de dato asociado a un dominio concreto, y las filas sus distintas instancias. El cuadro 1.2 muestra un ejemplo de esta representación sobre la asistencia de bañistas a una piscina y la temperatura ambiente en un instante de tiempo.

Fecha	**Temperatura ambiente (Cº)**	**Número de bañistas**
10/08/2019 17:15:00	30	45
11/08/2019 17:15:00	34	47
12/08/2019 17:15:00	34	49
13/08/2019 17:15:00	34	46
14/08/2019 17:15:00	27	36
15/08/2019 17:15:00	21	12

Cuadro 1.2: Ejemplo de un *dataset* de series de tiempo

El número de filas, y por tanto de elementos, del conjunto de datos debe ser lo **suficientemente amplio** como para tener una muestra realista del evento que se está estudiando. Por ejemplo, si se registran datos cada quince minutos, será suficiente con contar con registros de varios días. Sin embargo, si la toma de datos es de una vez por día, será necesario reunir varios meses de muestras para que sea significativo. Además de la cantidad de elementos de la que disponemos, también es necesario realizar una **curación y limpieza de los datos** de los que se disponen, asegurándose de que no haya contenido erróneo, vacío o repetido. A modo ilustrativo, el cuadro 1.3 presenta un *dataset* no válido sobre la actividad de un usuario en redes sociales.

Fecha	**Número de publicaciones**
20/08/2019 23:59:59	4
21/08/2019 23:59:59	0
22/08/2019 23:59:59	9
NULL	1
24/08/2019 23:59:59	5
25/08/2019 23:59:59	**NULL**
25/08/2019 23:59:59	3
25/08/2019 23:59:59	**-1**

Cuadro 1.3: Ejemplo de un conjunto de datos no curado

Cuando las distintas variables observadas vienen indexadas por una columna que define una fecha, como en los ejemplos mostrados anteriormente, se conoce como datos de **series de tiempo** o por su anglicismo *time series data*.

Una vez se cuentan con los datos correctamente dispuestos, el siguiente paso que tiene lugar es la aplicación de un algoritmo sobre dichos valores para **identificar patrones en la muestra de datos**, fase que habitualmente se conoce como **entrenamiento**. El resultado de la ejecución de estas instrucciones es la obtención de un **modelo de regresión** para el aprendizaje automático, sobre el que posteriormente se generarán valores a partir de nuevos datos de entrada.

Suponiendo el ejemplo inicial sobre la asistencia de bañistas a una piscina, la aplicación de un modelo de regresión, entrenado con registros de datos históricos con frecuencia de una hora, podría proporcionarnos una **predicción de cuál será la próxima ocupación** en base a la temperatura ambiente esperada.

1.3. Identificación del problema y objetivo del proyecto

A fecha de presentación de este documento, el **liderazgo de Python como plataforma para la implementación de *Machine Learning*** es indudable, debido tanto a la activa comunidad de usuarios y desarrolladores que lo soportan como a la gran cantidad de librerías que hay disponibles. Sin embargo, este lenguaje de programación no suele ser el habitual a la hora de elaborar sistemas de gestión en la empresa, sino que están arraigados otros entornos de desarrollo como .NET de Microsoft, como ocurre en el caso de IDbox.

Como parte de mejora de las características del servicio, IDbox considera la posibilidad de **invocar código Python específico de *Machine Learning* desde su desarrollo en lenguaje C#**. Es decir, se desea la **interoperabilidad entre estos dos lenguajes**, cuyas características son dispares, de manera que la obtención del resultado final en .NET a partir de la entrada en Python sea transparente. De este modo, el usuario final debe ser capaz de invocar operaciones de aprendizaje automático con un **gran volumen de datos** estructurados a través de IDbox sin que esta incompatibilidad de tecnologías afecte al funcionamiento de cada caso de uso.

Conociendo que Python expone un ejecutable que puede ser invocado para la ejecución de código compatible con su sintaxis, una primera solución que puede ser fácilmente contemplada es la llamada desde C# al intérprete `python.exe` con el código fuente como uno de sus argumentos. Esta primera aproximación es la que se muestra a continuación.

```
public void PythonExecution(string args, ref result, ref errors)
{
        ProcessStartInfo start = new ProcessStartInfo();
        start.FileName = "PYTHON_EXE_PATH";
        start.Arguments = args;
        start.UseShellExecute = false;
        start.RedirectStandardOutput = true;
        start.RedirectStandardError = true;
        using (Process process = Process.Start(start))
        {
                using (StreamReader reader = process.StandardOutput)
                {
                        string result = reader.ReadToEnd();
                        string errors = process.StandardError.ReadToEnd();
                }
        }
}
```

Sin embargo, aunque esta implementación podría llegar a funcionar en aquellos programas de Python simples y en casos en los que no se produce ningún error, cuando se plantean los problemas que pueden darse en un flujo alternativo al ideal, rápidamente se demuestra que **esta primera aproximación es insuficiente**:

- **Desbordamiento de la capacidad de la máquina** por el inicio de más procesos Python de los que pueden procesarse.

- Imposibilidad para **detectar y tratar errores en tiempo de ejecución**.
- Datos estructurados de entrada en un **formato no comprensible** por el programa de Python a invocar.
- **Desconocimiento del estado** en el que se encuentra una ejecución: ¿cómo distinguimos una ejecución correcta, que pueda tomar un largo periodo de tiempo, de un bucle infinito producido por error?
- **Restricciones a la hora de elevar procesos** por parte de la máquina o del servidor web que alberga la lógica de negocio.
- **Realización de trabajo irrelevante** debido a la repetición de invocaciones: varias llamadas de la misma operación podrían almacenarse temporalmente.
- **Dependencia del entorno de ejecución** del intérprete en la misma máquina que la lógica de negocio: aún con invocación remota de objetos en otro servidor sería necesario al menos un cliente Python.

Debido a las limitaciones anteriormente expuestas, la solución requiere de un **desarrollo a medida más complejo** que garantice: **el control de la ejecución, de errores y del estado, abordar la comunicación entre las distintas entidades, y que pueda ser integrado con IDbox**. Este artefacto *software* será un **motor de aprendizaje automático** que, además de todas aquellas opciones relativas a su mantenimiento y administración, deberá será capaz de resolver las operaciones de inteligencia artificial que se desean soportar:

1. **Generación de un modelo**: en base a una colección de datos podrá entrenarse un modelo de regresión y almacenarse para su posterior uso.
2. **Generación de predicciones**: a partir de un modelo previamente entrenado será posible obtener una predicción de una colección de datos proporcionada.

1.4. Organización de la memoria

El resultado del trabajo analizado, diseñado e implementado se muestra a lo largo de los próximos apartados de la memoria organizados de la siguiente forma:

1. En primer lugar, y tras conocer todos los conceptos fundamentales en esta introducción, el segundo capítulo **analiza detalladamente los requisitos que forman parte del proyecto**, donde se describen los casos de uso y se justifica la elección de las herramientas a utilizar para la ejecución de la propuesta de solución al problema.
2. Posteriormente, el tercer capítulo describe paso a paso el **diseño realizado así como los detalles de su implementación**, tanto de la creación del motor de aprendizaje automático como de su integración en IDbox. Para su validación, el cuarto capítulo aborda la **descripción y resultado de las distintas pruebas incorporadas**.

3. Con todo el sistema diseñado, implementado, integrado y probado, el quinto capítulo muestra **la explotación del sistema desde el punto de vista de los usuarios que forman parte de la solución al problema**; finalizando con las **conclusiones sobre el proyecto y futuras líneas de trabajo identificadas** en el sexto capítulo.

Capítulo 2

Análisis de requisitos

Una vez identificado el problema y justificada la necesidad de contar con un motor de Machine Learning que dé soporte a las dos operaciones de aprendizaje automático, a lo largo del presente apartado se analiza en detalle **la interacción entre los distintos actores y casos de uso** que esquemáticamente se muestra en la figura 2.1. La realización de esta fase sentará las **bases necesarias para abordar correctamente el posterior diseño e implementación**.

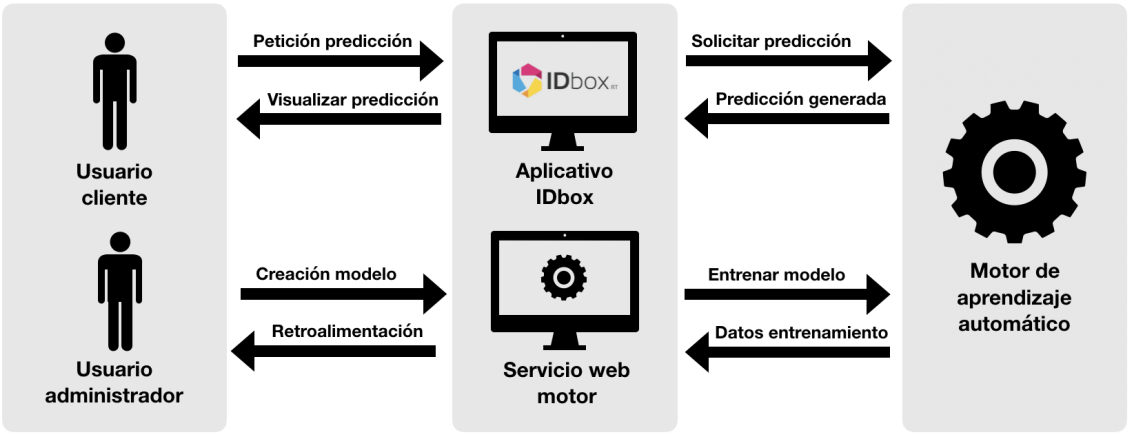

Figura 2.1: Representación esquemática de los requisitos del sistema

Por un lado se cuenta con el cliente, que va a poder obtener predicciones de sus datos directamente desde IDbox, pues la ejecución de esta operación **no requiere de ningún conocimiento sobre el dominio de aprendizaje automático**. Sin embargo, generar un modelo de manera adecuada **requiere de conocimiento sobre los algoritmos que se están aplicando y del proceso de entrenamiento**. Es por ello que, esta última operación será llevada a cabo únicamente por personal cualificado para cada necesidad del cliente, esto es, el administrador.

Como una primera aproximación y debido a este razonamiento, la fase de **entrenamiento del modelo no forma parte de una de las acciones disponibles desde la web de IDbox**, sino como una llamada al motor de Machine Learning.

2.1. Especificación de casos de uso

Tras una vista esquemática de los requisitos del sistema, la figura 2.2 muestra los casos de uso de la aplicación **organizados en dos grupos correspondientes a cada subsistema que integra la solución global**.

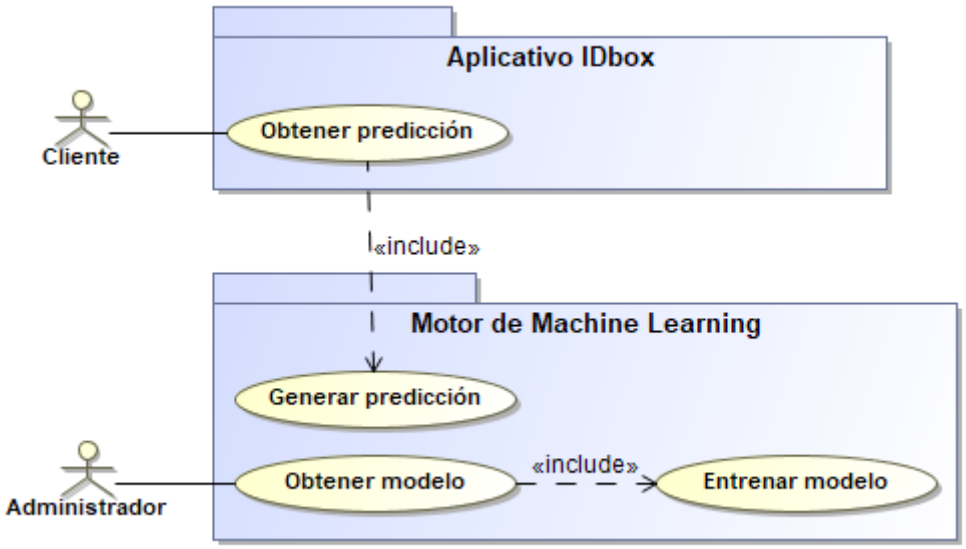

Figura 2.2: Diagrama de casos de uso

En la realización de este trabajo fin de grado, cuando se hace referencia a la obtención de predicciones **se supone un modelo de regresión ya entrenado para históricos de series de datos de una variable**.

A continuación, los cuadros 2.1 y 2.2 detallan los dos casos de uso de alto nivel obtenidos como resultado del análisis de requisitos: **"Obtener predicción" y "Obtener modelo"** respectivamente. Para cada uno se muestra su descripción de alto nivel, los actores involucrados, la secuencia detallada de acciones para el caso normal y el caso en el que se producen excepciones, así como su precondición y post-condición. Debido a que los casos de uso "Generar predicción" y "Entrenar modelo" se corresponden con los algoritmos internos de aprendizaje automático, éstos no han sido especificados.

Nombre	Obtener predicción.
Descripción	Un cliente obtiene una predicción de una o más variables seleccionadas para ello.
Actores	Cliente (primario). Motor de aprendizaje automático (secundario).
Precondición	El cliente se encuentra autenticado en el sistema, en el que ha seleccionado un intervalo de tiempo correcto y una o más variables que contienen datos históricos.
Secuencia normal	1. El cliente solicita al sistema obtener una predicción de una o más variables. 2. Para cada una de las variables marcadas por el cliente: 2.1. El sistema recupera los datos históricos de la variable en el intervalo indicado. 2.2. El sistema procesa los datos históricos recuperados para su curado. 2.3. El sistema transforma los datos del formato de recuperación al formato empleado en la transmisión. 2.4. El sistema envía la petición al motor. 2.5. Include "Generar predicción". 2.6. El sistema captura la respuesta de la petición efectuada. 3. El sistema presenta gráficamente el resultado: 3.1. El sistema pinta la visualización de los datos históricos como gráfica de tendencia. 3.2. El sistema añade a la gráfica de tendencia los puntos de predicción.
Postcondicion	El usuario ha obtenido una predicción, basada en el modelo por defecto, de las variables elegidas por éste.
Secuencia alternativa	3.2.a. El sistema notifica al usuario con un mensaje de error informando sobre la imposibilidad de obtener la predicción.

Cuadro 2.1: Especificación del caso de uso "Obtener predicción"

Nombre	Obtener modelo.
Descripción	Un administrador entrena un modelo de regresión con datos históricos de series de tiempo de una variable.
Actores	Administrador (primario).
Precondición	El administrador cuenta con acceso a la API del motor de aprendizaje automático y con los datos históricos de entrenamiento correctamente preparados y formateados.
Secuencia normal	1. El administrador solicita a la API del motor de aprendizaje automático entrenar un nuevo modelo: 1.1. El administrador proporciona el nombre del modelo que quiere entrenar. 1.2. El administrador proporciona los datos históricos correctamente preparados y formateados. 2. El sistema crea la petición: 2.1. El sistema da de alta la petición a la que le asigna un identificador único. 2.2. El sistema retorna como respuesta el identificador único asignado a la petición. 2.3. El sistema muestra el progreso de la petición en el *dashboard* de administración. 3. El sistema procesa la petición: 3.1. El sistema asigna la petición a un *worker* disponible. 3.2. Include "Entrenar modelo". 3.3. El sistema almacena el modelo entrenado con un identificador. 3.4. El sistema proporciona el identificador único de modelo. 4. El administrador obtiene el modelo: 4.1. El administrador comprueba que se haya completado la petición. 4.2. El administrador obtiene el identificador asociado al nuevo modelo creado.
Postcondicion	El administrador dispone del identificador único asociado al modelo que se ha entrenado para futuras operaciones de predicción.
Secuencia alternativa	4.2.a. El administrador obtiene un mensaje de error como resultado de no haber podido entrenarse el modelo.

Cuadro 2.2: Especificación del caso de uso "Obtener modelo"

2.2. Planteamiento inicial de la solución

Como representan las figuras 2.3 y 2.4, esta solución podría codificarse en cualquiera de los dos lenguajes, esto es, el control podría incorporarse **como parte de la lógica de negocio de IDbox en el entorno .NET o como parte del dominio de ejecución del código que se quiere ejecutar en Python**.

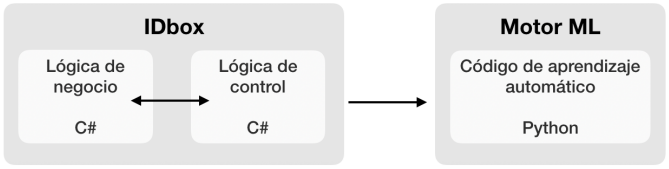

Figura 2.3: Lógica de control en .NET

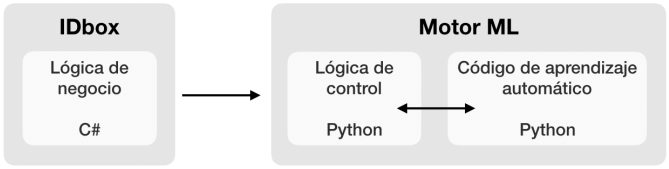

Figura 2.4: Lógica de control en Python

Para cada uno de los entornos, el cuadro 2.3 muestra un estudio de las características asociadas a los requisitos solicitados.

Requisito	**Control en .NET**	**Control en Python**
Control de errores	Limitado: obtención del mensaje de error	Completo: tratamiento del error de manera nativa
Estado de la ejecución	Limitado: vista caja negra	Completo: vista caja blanca
Sentido de la comunicación	Limitado: lectura de mensajes por consola	Completo: bidireccional, envío y recepción
Necesidad de privilegios	Dependiente de la configuración del servidor	Ninguno
Dominio de ejecución	Limitado al servidor que contiene el intérprete	Posibilidad de distribución
Explotación del motor	Limitado al lenguaje de control (C# en este caso)	Multiplataforma

Cuadro 2.3: Posibilidades que ofrecen las distintas soluciones

En base a este análisis, y debido a que la lógica de inteligencia artificial que se desea integrar se desarrolla en Python, **el lenguaje del motor a implementar será Python** con la finalidad de lograr una ejecución nativa en la que se disponga de una administración y control total. El proporcionado de datos y la obtención de resultados se realizarán siguiendo un protocolo estándar que pueda ser procesado por cualquier otro lenguaje, que para este caso en concreto será C#.

2.3. Estudio sobre las implementaciones de la especificación de Python

Al igual que ocurre con el resto de lenguajes, Python cuenta con una **documentación de referencia** acerca de su sintaxis y comportamiento, esto es, su interfaz del lenguaje [5]. Sin embargo, las implementaciones que realmente codifican esta descripción son extensas [6], motivo por el que se ha realizado un minucioso estudio sobre las características de las más relevantes, con el objetivo de elegir la más adecuada para utilizar en el proyecto.

Por un lado, se encuentran aquellas **implementaciones que exponen un intérprete** que ofrece el servicio de lectura e interpretación del código Python: CPython [7], Stackless Python [8], Pypy [9] o MicroPython [10] / CircuitPython [11]. Por otro lado, se sitúan las **implementaciones de compiladores cruzados** o en su anglicismo *cross compiler*, es decir, traductores de lenguaje de una a otra plataforma: IronPython [12], Python for .NET [13] o Jython [14]. En el cuadro 2.4 se muestra un resumen de dicho trabajo, que da paso a la conclusión sobre la implementación elegida.

Implementación	Versión	Características
CPython	3.7.4 (8 de julio de 2019)	Implementación de referencia escrita en C y Python con soporte multiplataforma y 64 bits.
Stackless Python	3.7.4 (4 de agosto 2019)	Bifurcación de la implementación de referencia, con la inclusión de *microthreads* que mejora el rendimiento de los *threads* de sistema operativo.
Pypy	3.5 (8 de febrero 2019) o 3.6 beta (16 abril 2019)	Implementación escrita en Python y RPython para incluir compilación en tiempo de ejecución y lograr un mejor rendimiento frente a la opción de referencia. Multiplataforma, pero sin soporte de 64bit en Windows.
IronPython	2.7 (9 de octubre de 2018) 3.x (en desarrollo, no disponible para su uso)	Implementado en C# permite usar librerías de .NET Framework y Python, con compilación en tiempo de ejecución y uso de la máquina virtual de .NET. Incorpora soporte de 64bit.
Python for .NET	3.7 (3 de mayo 2019)	Funcionamiento análogo a IronPython.
MicroPython / CircuitPython	Parte de la sintaxis de la versión 3.X	Versión ligera (no incluye toda la especificación) implementada en C y optimizada para microcontroladores y sistemas empotrados.
Jython	2.7 (noviembre 2017)	Implementación en Java que corre sobre su máquina virtual y permite utilizar librerías de dicho lenguaje.

Cuadro 2.4: Comparativa de implementaciones del intérprete de Python

Tras este estudio, se ha determinado que las únicas **opciones viables son las implementaciones de CPython y Stackless Python**, ya que integran completamente, y en su versión más reciente, todas las características de la documentación de referencia. Aunque el resto de alternativas incorporan algún tipo de funcionalidad ventajosa, para ello se sacrifica la compatibilidad, completitud y la exactitud de la especificación original, requisitos fundamentales en las operaciones de aprendizaje automático que utilizan librerías e instrucciones concretas del ámbito matemático y estadístico.

Tras sopesar la posible utilidad que ofrecen las mejoras de Stackless Python, y aunque inicialmente fue considerado el uso de sus *microthreads*, finalmente se ha decidido **emplear CPython** debido a que su implementación satisface todos los requisitos.

2.4. Elección de las herramientas de trabajo en entorno Windows

Antes de comenzar con el desarrollo, y tras una fase de investigación y análisis, es imprescindible definir las herramientas de trabajo que van a ser utilizadas en el entorno Windows para cumplir con los requisitos solicitados. A continuación se lista cada una de ellas, mencionando su utilidad y motivos de su elección.

- **Visual Studio y NuGet**: desde su lanzamiento en los años 90, se trata del editor más consolidado a la hora de desarrollar en entornos .NET, como ocurre con IDbox y su empleo de C#. Debido a este motivo y por coherencia con las herramientas utilizadas en el contexto de la empresa ha sido el elegido para la parte de implementación con IDbox, junto con el gestor de paquetes NuGet para la utilización y mantenimiento de dependencias.

- **Visual Studio Code**: este editor de código de nueva generación creado por Microsoft, ofrece todas las funcionalidades esperadas de un IDE tradicional y la posibilidad de desarrollar en varios lenguajes al mismo tiempo. Esto representa una clara ventaja para un proyecto como el planteado en este documento, donde será necesario escribir código Python, HTML, Javascript, CSS, entre otros.

- **Git**: tanto para el desarrollo del código como para la elaboración de la memoria en LaTeXse ha definido Git como software para el control de versiones, a través de la integración existente con los IDEs en el primer caso o a través de línea de comandos en el segundo.

- **Interfaz de línea de comandos de Windows**: mediante los comandos ofrecidos por esta interfaz se realizarán invocaciones parciales de código así como peticiones de red para la realización de pruebas puntuales durante el desarrollo.

- **Herramienta para desarrolladores de Chrome**: será indispensable para obtener detalles avanzados sobre el funcionamiento interno de la parte de software invocada desde el navegador: monitorización de la actividad de red, renderizado de los elementos de la interfaz, información de rendimiento, entre otros.

- **Jira**: antes de comenzar a desarrollar código, es imprescindible determinar cuáles van a ser las tareas a desarrollar y cuál es la estimación de tiempo que puede tomar cada una. De esta manera no solo se obtiene una carga de trabajo realista, sino que además es posible observar cuál es el estado durante el progreso. Jira, una de las herramientas más populares para la gestión de proyectos a nivel empresarial, es la que se ha utilizado para este propósito tanto para definir el alcance como para observar de manera clara la evolución a lo largo del tiempo.

2.5. Metodología de desarrollo

Para limitar el alcance del proyecto a las horas de dedicación del trabajo fin de grado y poder realizar su seguimiento, tras la fase de análisis inicial se identificaron las distintas subtareas que lo conforman, tal y como muestra la figura 2.5. El desglose de cada una implica **detallar una descripción de la actividad** así como **proporcionar una estimación del tiempo que se prevé que pueda tomar su realización**.

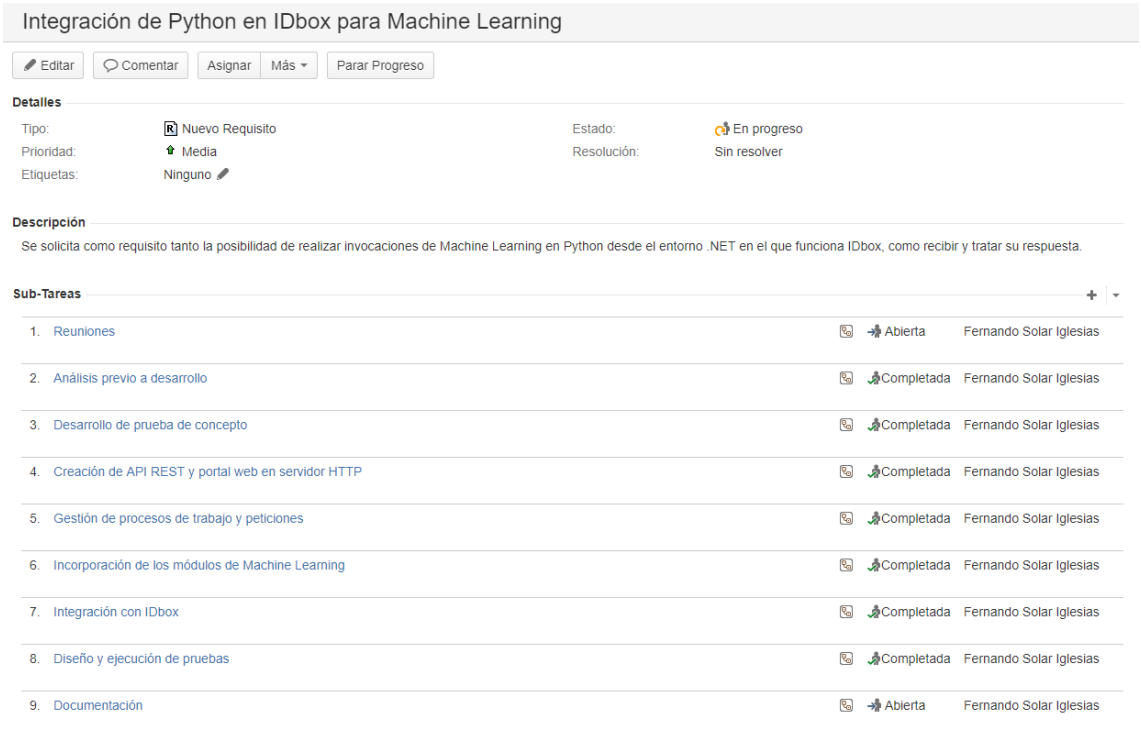

Figura 2.5: Desglose de tareas en Jira

Con el plan de trabajo planificado, el siguiente paso que tiene lugar es su ejecución. Debido a la rigidez que exige el modelo tradicional de desarrollo de *software* en cascada, en el que se definen unas etapas que se van completando a medida que termina la anterior, se ha optado por seguir un **modelo iterativo e incremental**. En este marco de trabajo, para cada iteración del producto se obtiene una **versión funcional del artefacto *software* sobre la que obtener retroalimentación de las partes interesadas**, de cara a seguir mejorando sus características hasta lograr el estado final deseado.

A modo ilustrativo de los incrementos, en un primer lugar se realizó la comunicación del motor con el exterior mediante el empleo de sockets por consola de comandos, que **posteriormente se refinó a la transferencia de mensajes HTTP sin estado**. De igual modo, la operación que inicialmente soportaba el motor como demostración era el cuadrado de un número proporcionado como entrada, siendo **reemplazado posteriormente por las operaciones reales de aprendizaje automático**.

Capítulo 3

Diseño e implementación de la solución

A lo largo de este capítulo se detalla el diseño de la solución para resolver el problema inicialmente planteado. El desarrollo se ha dividido en **dos partes diferenciadas** que interaccionan entre sí, como muestra la figura 3.1, utilizando un protocolo de comunicación que ambos artefactos software pueden procesar: **el intercambio de mensajes HTTP sobre TCP/IP**.

- **Creación del motor en lenguaje Python**: desarrollo del servidor encargado de ejecutar y controlar las operaciones de aprendizaje automático invocadas de manera externa, garantizado los requisitos señalados en la fase de análisis. Para la introducción y obtención de datos se expone tanto una API REST como un portal web, por lo que también se emplea HTML, CSS y Javascript como parte de los lenguajes de desarrollo para cubrir la parte visual de la aplicación.
- **Integración con la lógica de negocio de IDbox en lenguaje C#**: tomando como base el último estado de desarrollo de IDbox, se realizan las modificaciones e inserciones de código necesarias para que la herramienta de inteligencia operacional haga uso del nuevo motor creado. Gracias a esta integración, el usuario final podrá ver satisfechos los casos de uso identificados al comienzo.

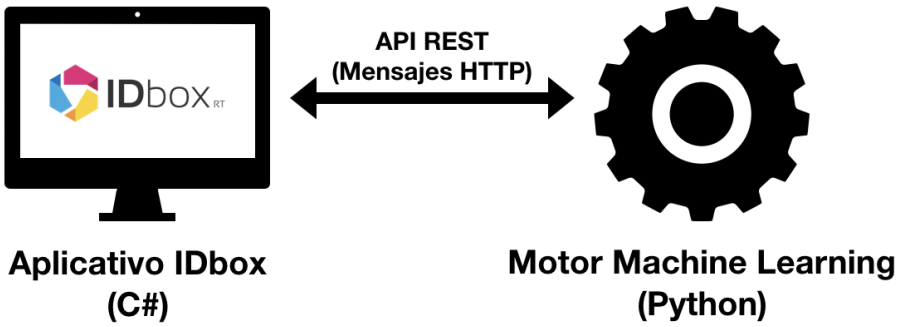

Figura 3.1: Interacción entre ambos artefactos software

Los mensajes HTTP **encapsulan a su vez un texto en formato JSON**, siglas del anglicismo *JavaScript Object Notation*, una especificación diseñada para el intercambio de datos cuyo uso se encuentra muy extendido a fecha de publicación

de este documento. Uno de los motivos por los que se prefiere su uso frente a alternativas propuestas con anterioridad, como XML (Extensible Markup Language), es su sencilla sintaxis limitada por los caracteres de corchete en el caso de listas de elementos o llaves en el caso de representaciones de tipo clave - valor. Estos mensajes pueden contener números racionales, o cadenas de texto delimitadas por el caracter de comillas.

- **Ejemplo de una lista formada por números enteros**: [-4, 0, -3, 2, 1, 1]
- **Ejemplo de un objeto coche**:

```
{
"matricula": "0000ABC",
"num_modelo": 37,
"id_propietarios": [1023447, 2343234]
}
```

Para la identificación de cada instancia de una entidad se utilizan **identificaciones únicos generados automáticamente** en dos formatos: numérico formado por un valor entero natural (ID), o alfanuméricos formados por números naturales y letras que se ven delimitados por guiones (GUID). Estas nomenclaturas, identificador e identificador único global respectivamente, son utilizadas a lo largo del diseño a la hora de identificar instancias concretas.

- **Ejemplo ID**: 23535
- **Ejemplo GUID**: eb4e38dc-04a4-4a89-9f70-0fc786f32fe3

3.1. Creación del motor: implementación en Python

La estructura del proyecto relativo al motor se ha construido en base a los conceptos del *Domain Driven Design*, esto es, el diseño guiado por el dominio: **objetos de valor y entidades** [15].

3.1.1. Objetos de valor

Como parte de los **objetos de valor**, o por su anglicismo *value object*, se encuentran todas las estructuras de datos utilizadas por la aplicación que definen el formato a utilizar para cada dato y los atributos que describen cada objeto. En lugar de utilizar estructuras genéricas como los vectores para modelar los conceptos del sistema, de esta manera se busca maximizar la integridad de los datos, es decir, que sean correctos y fiables.

3.1.1.1. Enumerados

Los **enumerados a nivel de aplicación** definen estados por los que pueden progresar, desde su creación hasta su eliminación, las dos principales entidades del motor: las tareas y los workers. En el proyecto se han diseñado dos tipos, mostrados en la figura 3.2 y descritos a continuación.

- **TaskStatuses**: define los estados en los que puede estar una tarea a lo largo de su ciclo de vida. En primer lugar, una tarea es creada recibiendo un identificador único alfanumérico, pasando a estar posteriormente asignada a un worker en concreto. Cuando realmente inicia su ejecución, la tarea se encuentra en progreso, cuyo desenlace puede ser su completado mediante la devolución de un resultado, o en caso contrario en estado de cancelada.

- **WorkerStatuses**: define los estados en los que se puede encontrar un worker a lo largo de su ciclo de vida. Tras ser correctamente inicializado, el worker pasa a estar disponible para aceptar carga de trabajo, situación a la que regresará después de estar ocupado con un trabajo en concreto. Como casos excepcionales y debido a una funcionamiento errático, el worker puede fallar inesperadamente o ser matado.

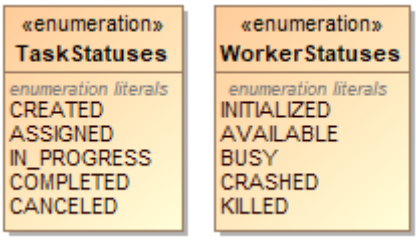

Figura 3.2: Enums de estado de las tareas y workers

3.1.1.2. Data classes

Las clases de datos hacen referencia a estructuras de valores con tipo que describen un objeto dentro de la aplicación. Tan solo contienen operaciones de acceso y de modificación de los datos, por lo que no incorporan lógica de negocio o funcionalidad avanzada. Como muestran las figuras 3.3 y 3.4, en el diseño del motor se han diferenciado dos tipos de *data classes* que se detallan a continuación.

Los **mensajes de colas de memoria** definen la estructura de los datos que contendrá esta estructura de datos de sincronización. El concepto de feedback hace referencia a una retroalimentación enviada por un worker que puede ser de tres tipos: de resultado, de progreso de la tarea, o información sobre el estado del worker.

- **ResultFeedback**: define un resultado obtenido tras la realización de operaciones asociadas a una tarea por parte del worker.

- **TaskFeedback**: los mensajes que siguen esta estructura proporcionan información sobre el estado en un instante determinado de una tarea en concreto que se encuentra en progreso en un worker. Este situación viene descrita por un porcentaje comprendido entre el 1 y el 100 y un código de estado definido a nivel de la aplicación.

- **WorkerFeedback**: para comunicar información en un instante determinado sobre un worker en concreto, como su id asignada por el sistema operativo (PID) o código de estado a nivel de aplicación, se define este tipo de dato.

- **ScheduledTask**: hace referencia a una tarea solicitada en un instante determinado por un cliente. Esta tarea programada, que se encuentra encolada pendiente de ser asignada por un worker, contiene la información de entrada que se usará posteriormente en el procesamiento.

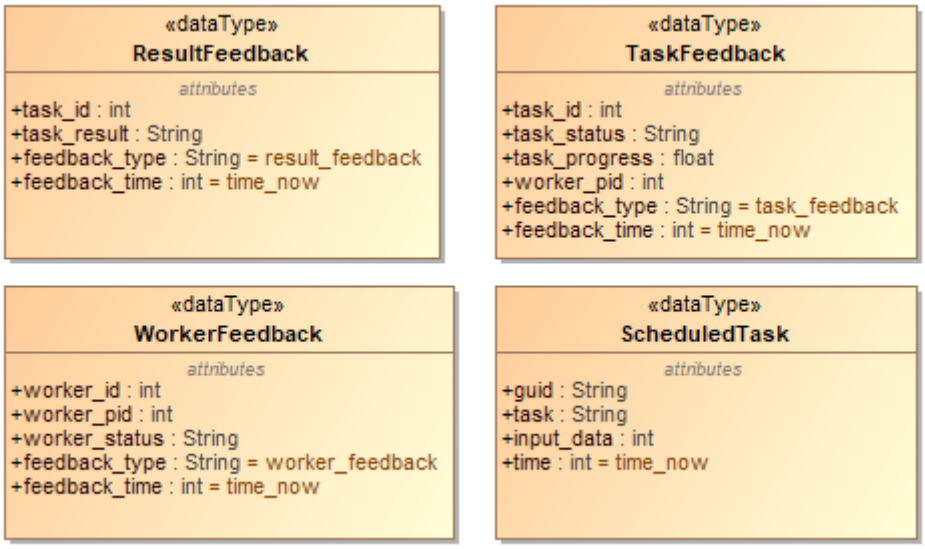

Figura 3.3: Data classes para las colas de memoria

Los **mensajes de servidor** modelan las estructuras de información sobre el motor que almacenará el servidor web hasta que un cliente los consuma.

- **TaskResponse**: define un resultado obtenido tras el procesamiento del motor, pendiente de ser consumido por un cliente.
- **WorkerStatus**: define el estado en el que se encuentra un worker, pendiente de ser consumido por un cliente.
- **TaskStatus**: define el progreso en el que se encuentra una tarea, pendiente de ser consumido por un cliente.

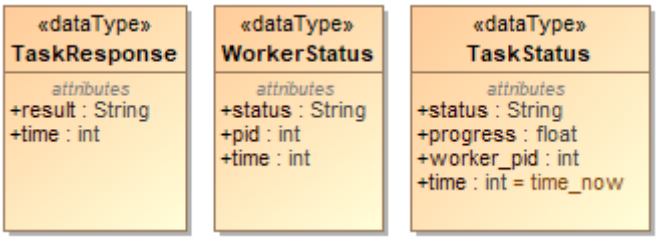

Figura 3.4: Data classes para los mensajes de servidor

3.1.2. Arquitectura: entidades y su sincronización

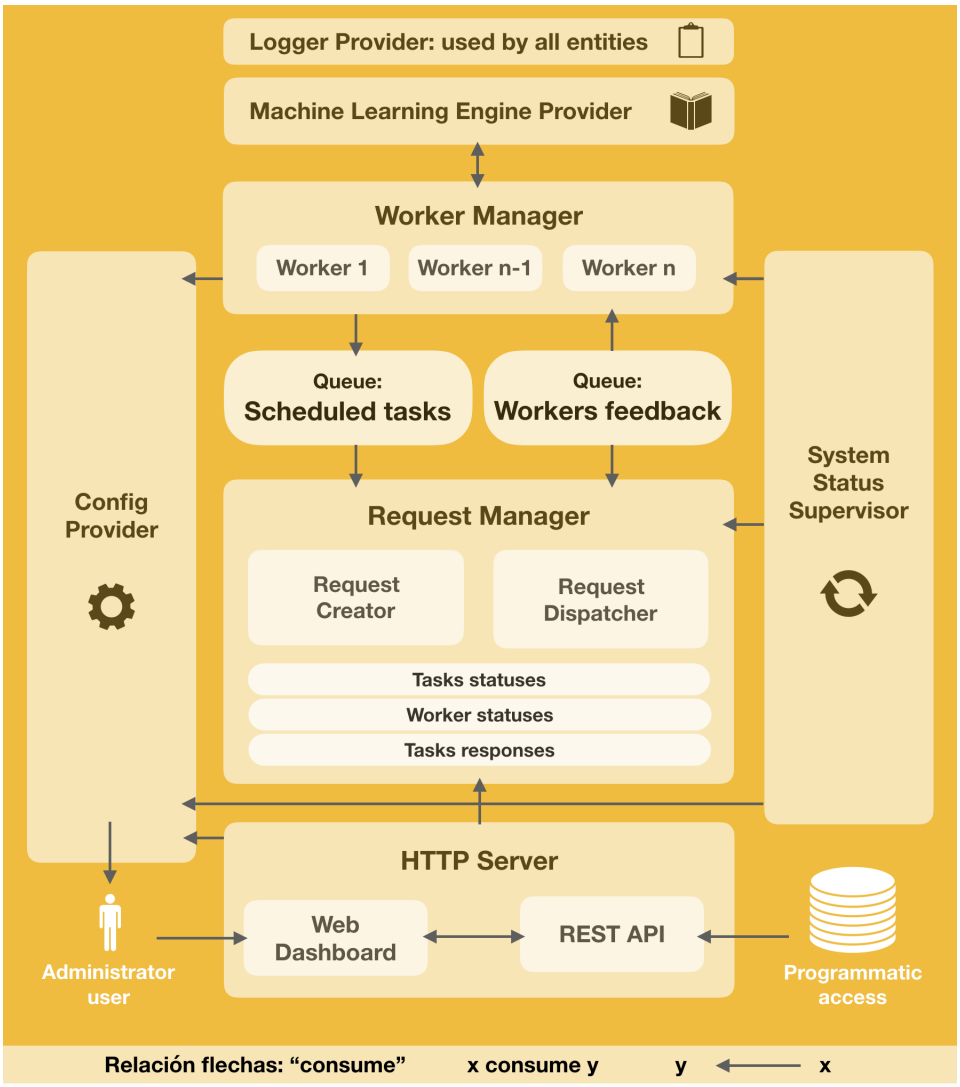

Figura 3.5: Arquitectura del motor de Machine Learning en Python

El motor de inteligencia artificial para aprendizaje automático cuenta con dos elementos fundamentales que forman parte del núcleo de la aplicación, que se enumeran y describen a continuación.

- **Worker**: proceso de trabajo aislado y totalmente independiente, constituido como un proceso a nivel de sistema operativo enlazado con las colas de memoria. Su entorno de ejecución contiene toda la especificación y recursos necesarios para ejecutar operaciones de aprendizaje automático.

    - **Interacción con el espacio de memoria**: entre las operaciones de comunicación con las colas de memoria que implementa se encuentran las funciones utilizadas por la implementación de aprendizaje automático para proporcionar información sobre el progreso de las operaciones. Por un lado, se tienen aquellas que dan soporte al envío de retroalimentación hacia el motor sobre el propio worker y la tarea que se encuentra ejecutando, y por otro lado la del envío del resultado final.

- **Lógica de funcionamiento**: en un flujo de funcionamiento correcto, la principal misión del worker es ofrecer un contexto para la ejecución de tareas de aprendizaje automático que se encuentren pendientes de procesar. Tras su instanciación e inicialización básica, el worker consume el próximo elemento disponible de la cola de memoria, de modo que se procesa y atiende esa petición, lanzando la ejecución de las instrucciones correspondientes. A partir de este punto, ante una situación excepcional no controlada que se produzca, el worker posee la capacidad para la cancelación de la tarea y posterior autodestrucción del proceso.

- **Tarea**: se tratan de un conjunto de instrucciones ejecutas por un worker que, invocadas por una petición del cliente al motor, satisfacen una operación. Cada uno de los casos de uso que se desean realizar vienen satisfechos por la creación de una tarea en concreto.

Para dar soporte al correcto funcionamiento de este núcleo de la aplicación, se cuenta con el `Configuration Provider`, el `System Status Supervisor`, el `Logger Provider`, el `Worker Manager`, el `Machine Learning Engine Provider`, y el `Request Manager`. Todos ellos son las **entidades que conforman el motor**, y a través de su disposición ordenada, que muestra la figura 3.5, se consigue un software modular cuyas partes se dedican exclusivamente a satisfacer una determinada funcionalidad, favoreciendo la legibilidad de código, la reutilización de componentes y la ampliación de sus capacidades.

Debido a que dos procesos independientes no pueden compartir información directamente, es necesario la utilización de un espacio compartido de memoria a nivel de sistema operativo para la comunicación segura entre varios productores y consumidores, es decir, entre los procesos de trabajo y las entidades del motor. Para cumplir con este requisito se ha elegido el uso de **colas de memoria** a través de las librerías de multiprocesamiento de Python, que encapsulan las llamadas correspondientes al tipo de sistema operativo que lo ejecuta. Estas colas de tipo FIFO, siglas de *First In First Out*: el primero en entrar es el primero en salir, proporcionan un **acceso seguro de alto nivel a los datos** mediante la encapsulación de mecanismos de sincronización de bajo nivel como pipes, locks y semáforos. Para ello **exponen tres operaciones principales**: añadir al final de la lista, consumir el primer elemento o conocer el tamaño en un instante [16, 17].

3.1.2.1. Configuration Provider

Todas aquellas variables de código cuyo valor podría resultar interesante de modificar en base al criterio del destino de implantación del software, se han definido como variables de clase. Además, con la finalidad de que esta **personalización pueda configurarse para cada cliente sin necesidad de volver a generar la aplicación**, se ha definido un archivo nombrado como `config.ini` cuyo contenido **contiene una instancia de una configuración concreta**.

A modo de ejemplo, la figura 3.6 muestra el fichero diseñado por defecto. Cada uno de los pares clave-valor se engloban dentro de un espacio de configuración definido entre corchetes, que en este caso hace referencia a las diferentes entidades del motor.

config.ini

```
[HealthManager]
# Health Manager will sleep this number of seconds before next check
CheckIntervalSeconds = 10
# Number of seconds to keep in memory a completed or canceled task before removal
RequestExpirationSeconds = 21600
# Number of seconds to consider a worker as dead
WorkerNotRespondingSeconds = 20

[WorkerManager]
# Number of workers that will be created on startup
NumStartupWorkers = 3

[RequestManager]
# Request Manager will sleep this number of seconds before next feedback check
CheckIntervalSeconds = 5

[WebServer]
# Webserver connection port
ConnectionPort = 5000
```

Figura 3.6: Visualización de la estructura del fichero de configuración

El **proveedor de configuración** es el encargado tanto del acceso a este fichero de pares clave y valor, como la posterior inicialización del conjunto de variables con los valores asignados. En caso de error por problemas en su lectura o no disponibilidad, se establece una configuración por defecto.

3.1.2.2. System Status Supervisor

Con el objetivo de que el funcionamiento de un programa informático a lo largo del tiempo sea el correcto, todo artefacto *software* requiere de la realización de tareas de supervisión de la ejecución para reducir las posibilidades de fallo a lo largo de su ciclo de vida. Con esta finalidad, el **supervisor de estado del sistema** es la entidad encargada de monitorizarlo y aplicar las siguientes operaciones:

- **Regeneración de workers fallidos**: aquellos procesos que están marcados con estado *crashed*, es decir, cuya ejecución ha fallado y no ha podido ser salvada, son suprimidos del sistema. Seguidamente, un nuevo proceso es levantado quedando a disposición del motor como reemplazo, de modo que la terminación fallida de varios workers no dejaría al motor sin unidades de procesamiento disponibles.
- **Limpieza de tareas completadas y canceladas**: las tareas que se encuentran un estado de finalización, y no han sido consumidas tras un tiempo de expiración fijado por la configuración, son monitorizadas para su eliminación. Esto incluye tanto a las tareas completadas con un resultado o aquellas que se encuentran canceladas.

- **Finalización de workers ausentes**: los procesos de trabajo envían periódicamente tramas de retroalimentación como prueba de vida, tanto del estado del propio worker como de la tarea que ejecuta. Conociendo esta premisa, esta circunstancia permite poder clasificar un worker como inactivo si el motor no recibe actualizaciones de éste tras un intervalo de tiempo configurable. Pasado dicho margen de tiempo, el proceso es terminado forzosamente para ser reemplazado por uno nuevo.

Sin esta entidad, los recursos generados por el motor podrían crecer de manera indefinida o detenerse su ejecución en un instante de tiempo, luego su implementación es imprescindible.

3.1.2.3. Logger Provider

El **proveedor de anotaciones** permite la escritura a fichero de trazas de mensajes dispuestos estratégicamente a lo largo del código. Para su clasificación, de acuerdo al módulo de logging de Python existen diferentes severidades ordenadas de menor a mayor:

- **Debug**: el nivel de detalle más bajo utilizado para el seguimiento de los bloques de instrucciones que se van ejecutando aunque no contengan errores.
- **Info**: avisos sobre la producción de comportamientos esperados.
- **Warning**: el artefacto software continua funcionando, pero se ha provocado una situación problemática que puede afectar ahora o en un futuro.
- **Critical**: nivel extremo de las severidades, el programa puede encontrar dificultades para continuar ejecutándose con normalidad.

A la hora de ejecutar el artefacto software, en caso de problemas o comportamientos no esperados, la lectura de este fichero de log resulta realmente útil ya que los propios mensajes anteriormente introducidos contienen información de interés para descubrir el origen del problema en tiempo de ejecución.

3.1.2.4. Request Manager: Creator y Dispatcher

El **gestor de peticiones** es el módulo encargado de recibir y enviar información entre los *workers* y la lógica del motor, a través de la interacción con las colas de memoria en sus dos operaciones principales: la inserción y consumo de elementos.

- **Request Creator**: recibe como entrada un código de operación, y los datos asociados a ésta, con la finalidad de dar de alta en el motor un nuevo trabajo. Para ello, el creador de peticiones comprueba que la actividad que desea iniciarse esté registrada en el sistema, creando en ese caso un identificador único que es devuelto al cliente. Es entonces cuando se añade a la cola de memoria de tareas programadas, y se asigna el estado *created* a la tarea.

- **Response Dispatcher**: el procesador de respuestas se trata del módulo encargado de consumir periódicamente la cola de memoria de retroalimentación de *workers*. Cuando una nueva trama es recibida, se analiza su tipo (retroalimentación de un worker, de una tarea o de respuesta) y se almacena en su contenedor temporal correspondiente a la espera de ser consumido por la API.

3.1.2.5. Worker Manager

El **gestor de workers** contiene la lógica para la elevación y terminación de procesos a nivel de sistema operativo: desde su creación, su posterior arrancado y espera a finalización. Para ello, se hace la uso de la librería de multiprocesamiento de Python que implementa las llamadas correspondientes para cada sistema operativo.

3.1.2.6. Machine Learning Engine Provider

La **lógica de aprendizaje automático** contiene todas las implementaciones que codifican las dos principales operaciones soportadas: el entrenamiento del modelo de regresión y la obtención de las predicciones asociadas a este.

Para facilitar la integración de desarrollos de Machine Learning codificados en Python se ha diseñado un marco de trabajo que pueda ser extendido a diferentes variantes de aprendizaje automático. La figura 3.7 muestra su estructura que se detalla a continuación.

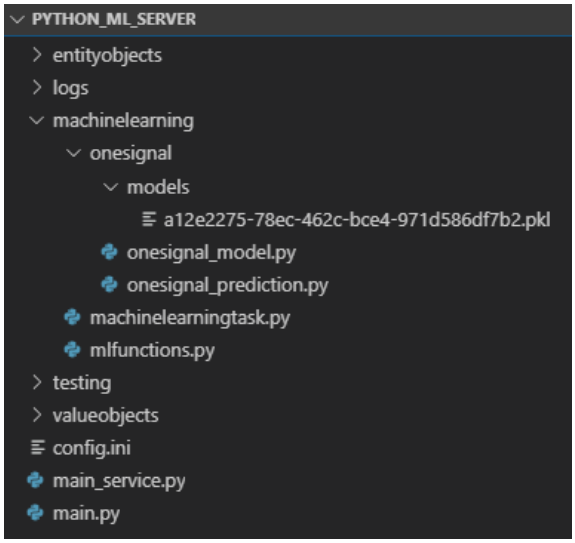

Figura 3.7: Visualización de la estructura del proveedor de aprendizaje automático

- **Machine Learning Task**: se trata de una clase esqueleto que será extendida por las distintas codificaciones que deseen integrarse y que encapsula internamente la lógica de comunicación con el worker, en la que se incluye el envío de la respuesta. De esta manera, el científico de datos solo debe preocuparse de incorporar su desarrollo como una sobreescritura de la operación run() que se expone.

- **ML-Functions**: para facilitar la reutilización de instrucciones y evitar la repetición de código, se ha establecido en la raíz de este módulo una librería de funciones comunes a varias implementaciones de aprendizaje automático. Para el caso desarrollado en este trabajo fin de grado, el fichero contiene aquellos algoritmos empleados en común en las operaciones de entrenamiento de modelo y obtención de predicción, pero podrían existir tantas librerías como fuese necesario.
- **ML-Folder**: cada uno de los tipos de aprendizaje automático que se deseen incorporar son integrados en su propio directorio, que contiene tanto la lógica de cada operación como un directorio en su interior con los ficheros serializados: en este caso los modelos entrenados.

3.1.2.7. HTTP Server: API REST y Web Dashboard

Para la comunicación de los mensajes JSON sobre HTTP en TCP/IP se expone un servicio REST, orientado al acceso programático, sobre el que puede realizarse la consulta o modificación de datos. La especificación completa del servicio con los recursos, direcciones, parámetros de consulta e interfaz uniforme se muestra en el cuadro 3.1.

Recurso	**URI**	**Query param**	**Interfaz uniforme**
Lista de workers	/api/workers	-	GET, 200 OK
Información de un worker	/api/workers/{workerid}	-	GET, 200 OK GET, 404 NOT FOUND
Lista de peticiones	/api/requests	-	GET, 200 OK POST, 201 CREATED POST, 400 BAD REQUEST
Estado de una petición	/api/requests/{guid}/status	-	GET, 200 OK GET, 404 NOT FOUND
Resultado de una petición	/api/requests/{guid}	-	GET, 200 OK GET, 404 NOT FOUND
Visualizar logs	/api/logs	severity	GET, 200 OK

Cuadro 3.1: Diseño de la API REST

Además de los códigos de respuesta específicos para cada recurso, el servidor retornará como respuesta "500: Internal server error" cuando un comportamiento no esperado se produzca en la lógica que implementa los recursos.

El soporte al diseño de esta interfaz viene dado por el **servidor HTTP apoyado en el framework Flask**, que permite vincular direcciones de recursos del servicio a la lógica de negocio mediante la decoración de los métodos Python. A modo de ejemplo, se muestra el decorador y cabecera para la operación de consulta y creación de peticiones:

```
@app.route("/api/requests", methods=['GET', 'POST'])
def api_requests():
```

La lógica de cada uno de estos métodos que dan soporte a la API se alimenta de los **contenedores temporales de la entidad Request Manager**.

En paralelo a este acceso programático, y para facilitar la administración del motor de aprendizaje automático, se ha desarrollado un **portal web que presenta gráficamente las operaciones de consulta a la API**. La figura 3.8 muestra su visualización a través de un navegador, cuyos detalles de implementación se especifican a continuación:

- **Maquetado**: sin la utilización de librerías de visualización, pero inspirado en las directrices de diseño de Material Design marcadas por Google, se ha codificado la parte visual del portal usando tecnologías HTML y CSS. Los componentes gráficos se ajustan en función del tamaño de la ventana del navegador, por lo que el diseño se adapta a diferentes resoluciones de pantalla posibles.
- **Lógica de funcionamiento**: con apoyo de la librería jQuery de Javascript, se ha codificado el controlador que da soporte tanto a la realización de solicitudes HTTP a la API como a la correcta disposición de la información. Además de mostrar el estado de los workers y peticiones, y la visualización de trazas de mensajes de la ejecución, se ofrece también cuál es el estado del servidor (verde para conectado o rojo para desconectado), así como la fecha de última carga de datos. Los valores se refrescan automáticamente según el periodo de tiempo fijado en la implementación, aunque el usuario puede realizar esta operación manualmente a través del botón ubicado en la parte superior derecha.

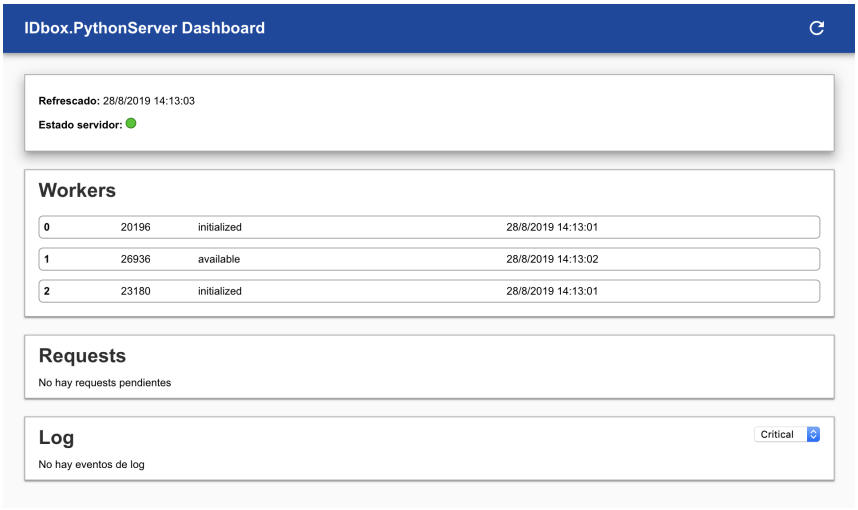

Figura 3.8: Vista del portal web para la monitorización del motor

Debido al gran volumen de datos que contiene el registro de la traza de mensajes de ejecución, se ha facilitado el filtrado según su severidad mediante un desplegable que muestra la figura 3.9. El nivel seleccionado se trata del valor que recibirá el parámetro de consulta "severity" del recurso "visualizar logs" de la API.

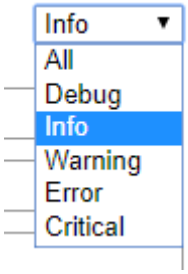

Figura 3.9: Filtrado de severidad de la traza de mensajes de ejecución

3.2. Integración con IDbox: implementación en C#

Para poder integrar el motor de aprendizaje automático con la lógica de negocio de IDbox es necesario **ampliar el diseño de la arquitectura web** con nuevas bibliotecas de interfaces y clases C#, tanto del *frontend* (parte que interacciona con el lado del cliente) como del *backend* (parte que corre contra las operaciones con datos). En las siguientes secciones se muestra, para cada apartado, el diseño y desarrollo realizado tras el estudio del último modelo de clases de la plataforma.

3.2.1. Frontend: interacción con el cliente web

El frontal no solo presenta gráficamente la interfaz de visualización al usuario, sino que además es el encargado de controlar y gestionar las acciones que éste realiza, por lo que **es necesario dar soporte desde este apartado a la generación de la predicción**. Sin embargo, durante los últimos años IDbox ha ido incorporando distintos mecanismos de inteligencia artificial codificados en otras tecnologías como C#. Es por ello que, a la hora de realizar esta integración, debe considerarse además su **coherencia y compatibilidad con otras lógicas alternativas** que muestran un comportamiento interno completamente diferente.

El diagrama de clases UML de la figura 3.10 muestra la jerarquía de clases completa del caso de uso, seguido de una descripción textual detallada.

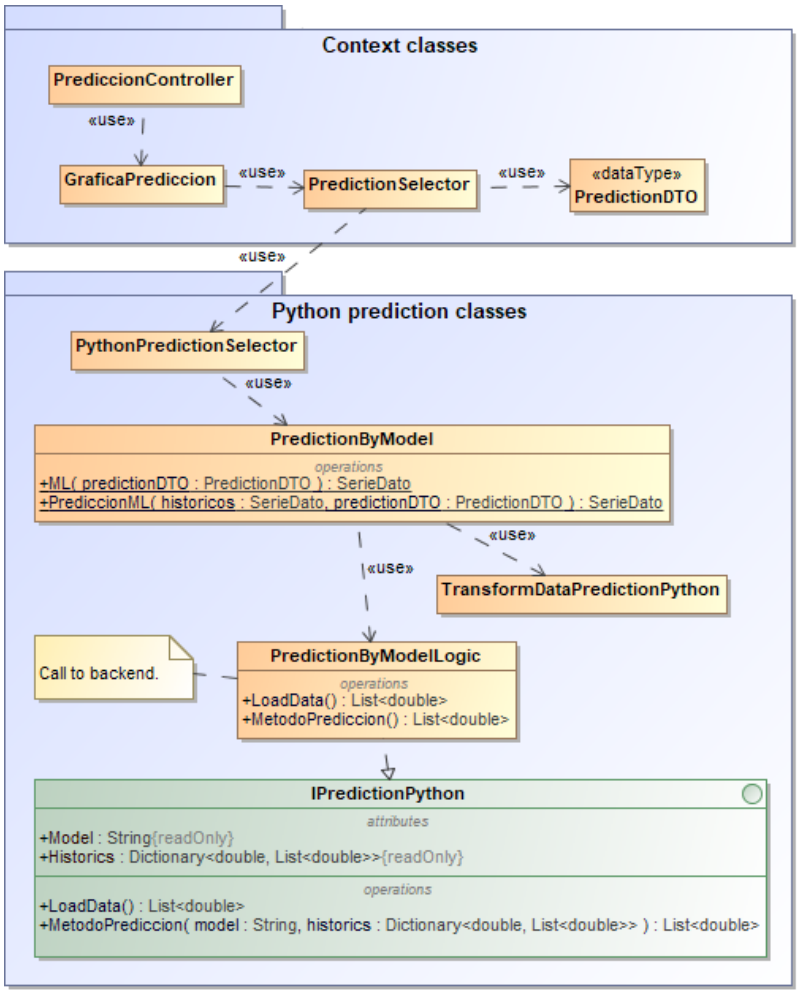

Figura 3.10: Diagrama de clases de la nueva funcionalidad en frontend

El paquete `Context classes` hace referencia a las clases del frontal relacionadas con el caso de uso, que ya se encontraban implementadas, pero que han sido modificadas para que **el nuevo tipo de predicción se vea soportado**. Una vez que se ha elegido esta clase de aprendizaje automático, se dispone del paquete `Python prediction classes`. Es aquí donde, con la interfaz `IPredictionPython`, se define un **contrato estándar para todas las posibles implementaciones que puedan añadirse**, cuya selección es realizada por el `PythonPredictionSelector` en base a la petición de predicción `PredictionDTO`. En concreto, **se implementa la predicción basada en un modelo** mediante `PredictionByModelLogic`, cuya lógica se delega al *backend* tras la necesaria **transformación y curación de los datos** implementada por `TransformDataPredictionPython`.

3.2.2. Backend: lógica de datos

En esta parte de la arquitectura tiene lugar el procesamiento de las órdenes recibidas desde el frontal. Es en este punto donde tiene lugar la comunicación con el motor de aprendizaje automático de Python, tanto para la petición de la operación correspondiente como para la obtención de resultados.

La figura 3.11 muestra el diagrama de clases UML correspondiente a la solución planteada y una descripción detallada de su implementación.

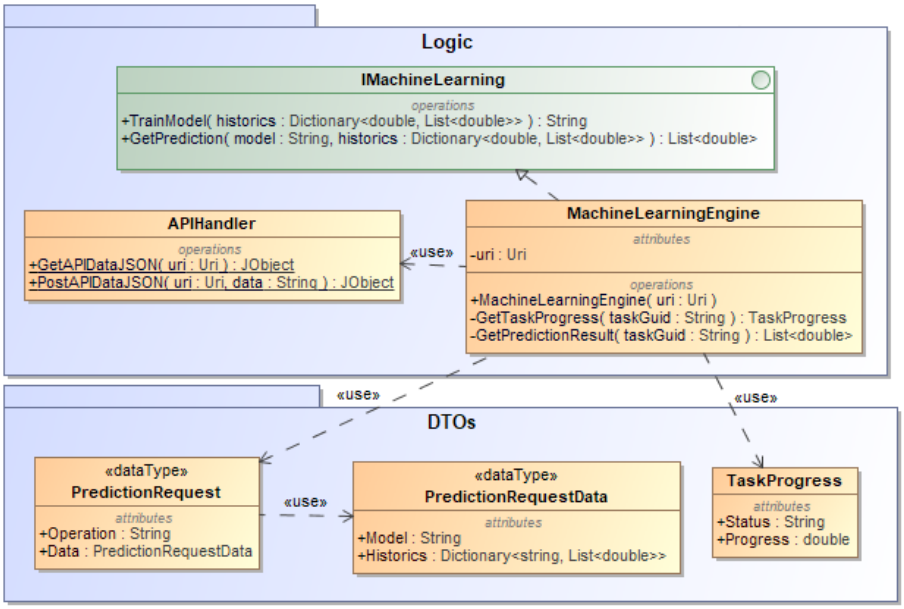

Figura 3.11: Diagrama de clases de la nueva funcionalidad en backend

Por un lado se encuentra la **lógica de comunicación** con el motor, a través del protocolo HTTP sobre TCP/IP, modelada en las siguientes clases:

- **APIHandler**: clase que hace uso del popular framework para el uso de JSON en .NET: Json.NET de Newtonsoft. Sus métodos realizan la operación GET o POST a la API, cuya localización de despliegue y datos de entrada (si se precisan) se proporcionan como argumentos. La respuesta viene dada como un objeto JObject propio del framework.
- **MachineLearningEngine**: encapsula la lógica de obtención de la predicción apoyándose en dos métodos auxiliares: obtención del progreso y respuesta de una tarea, que a su vez se ven soportados por el `APIHandler`. La obtención del resultado se obtiene mediante un mecanismo de encuesta al estado de la tarea, de modo que cuando es completada se consume su valor final para ser retornado al frontal.

Por otro lado se encuentran los **DTOs**, abreviatura del anglicismo *Data Transfer Object*, es decir, todos aquellas definiciones de objetos construidos para modelar en forma de estructura un intercambio de datos entre dos entidades. La finalidad de éstos es análoga a la descrita para los *data classes* de Python:

- **PredictionRequest y PredictionRequestData**: describen la estructura interna de la trama de petición que se convertirá a formato JSON.
- **TaskProgress**: describe el avance de una tarea, esto es, el estado y porcentaje de avance.

Capítulo 4

Diseño e implementación de pruebas

Cuando se trata de implementar un nuevo artefacto software, su desarrollo no termina con la implementación del diseño. La elaboración de un **conjunto de pruebas** y su **validación antes de la puesta en producción** es un paso indispensable al crear o actualizar una funcionalidad. En este caso se han aplicado tres niveles de prueba: **unitarias** para la validación automatizada de funciones concretas, de **integración** para la prueba automática de una agrupación de funcionalidad y de **aceptación** para la validación manual con el usuario.

4.1. Pruebas unitarias

Aunque Python incorpora una librería nativa para el desarrollo y ejecución de pruebas unitarias denominada unittest, el framework de terceros **pytest se ha convertido en el habitual** para resolver esta tarea. Esta aceptación viene en parte por su ritmo de actualización y añadido de características, y es que a diferencia del paquete integrado, nuevas versiones son liberadas periódicamente sin importar la versión de Python que se esté ejecutando.

Para su implementación, se ha creado **en la raíz del proyecto un directorio denominado tests** donde se almacenan tanto los objetos simulados como las propias clases que codifican las pruebas. Estas últimas comienzan por el prefijo "test", seguido con el nombre del componente que se valida y un identificador que describe el aspecto a probar. Como ejemplo, la prueba "test-machinelearningtask-execution" corresponde a la validación del componente que define una ejecución de una operación concreta de aprendizaje automático. De esta manera, a la hora de ejecutarse el proceso automático de pruebas, **pytest es capaz de identificarlas y posteriormente mostrar el resultado de cada una**. Esta retroalimentación es posible visualizarla mediante la salida por consola o gráficamente a través del propio entorno de desarrollo integrado como muestra la figura 4.1.

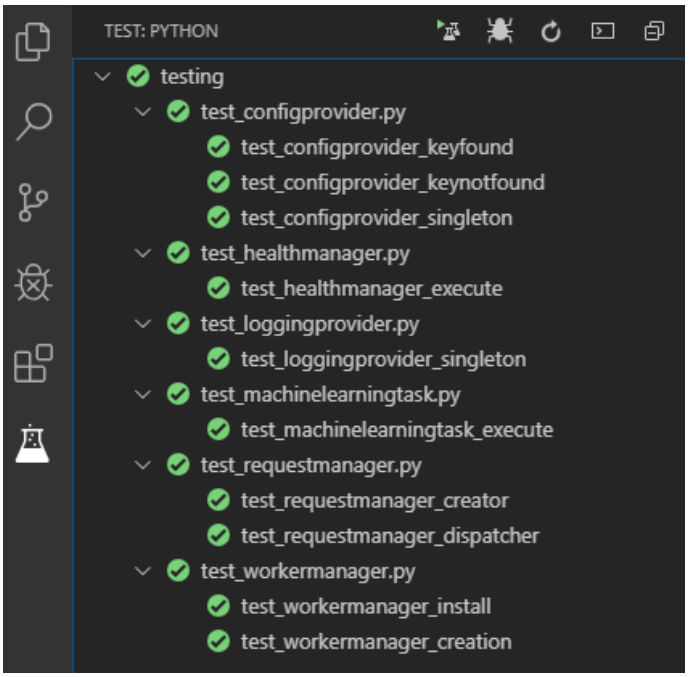

Figura 4.1: Visualización en Visual Studio de la ejecución del conjunto de pruebas

El listado de pruebas que se han aplicado se presenta a continuación junto a la descripción de cada una:

- **test-configprovider-keyfound**: de un fichero de configuración de prueba con valores conocidos, se comprueba que se obtiene el valor esperado para una variable en concreto.
- **test-configprovider-keynotfound**: de un fichero de configuración de prueba con valores conocidos, se comprueba que el acceso a una variable que no existe devuelve el valor configurado por defecto.
- **test-configprovider-singleton**: se invocan dos instancias del proveedor de configuración y se comprueba que ambas tienen la misma dirección de memoria, esto es, apuntan a la misma referencia.
- **test-requestmanager-creator**: se dispone de una cola de memoria de tareas programadas inicialmente vacía. Al solicitar una nueva tarea soportada, se comprueba que la cola contiene dicho elemento, cuya id es la misma que la retornada en el instante de creación, y que en los estados de tareas figura inscrita sin progreso.
- **test-requestmanager-dispatcher**: se dispone de una cola de memoria de retroalimentación de *workers* vacía, que se inicializa con una retroalimentación de cada tipo. Se comprueba que, tras ejecutar el *dispatcher*, se encuentran las tramas procesadas de *workers* y tareas con las id anteriormente establecidas.
- **test-healthmanager-execute**: dados unos estados ficitios de workers y tareas, se ejecuta el supervisor de estado del sistema y se comprueba que las listas de estados no contienen los elementos sujetos a su borrado.

- **test-loggingprovider-singleton**: se invocan dos instancias del proveedor de anotaciones y se comprueba que ambas tienen la misma dirección memoria, esto es, apuntan a la misma referencia.

- **test-machinelearningtask-execute**: se invoca, en un worker mock, una tarea de Machine Learning ficticia que implementa la interfaz original. Se comprueba que la comunicación entre ambos es correcta, esto es: el worker recibe el resultado esperado.

- **test-workermanager-install**: se comprueba, usando workers mock, que el gestor de workers realiza la operación de inicialización correctamente, esto es: el número de workers levantados es el esperado.

- **test-workermanager-creation**: se comprueba, usando workers mock, que el gestor de workers crea un nuevo worker y devuelve el identificador esperado.

Durante la ejecución de estas pruebas automatizadas, en lugar de crear instancias de las clases reales que dificultarían ser utilizadas para este objetivo, **se han utilizado objetos simulados** conocidos por su anglicismo *mock*. Estos implementan y heredan las mismas clases que las instancias reales, pudiendo probar así si el comportamiento es el esperado.

4.2. Pruebas de integración

Una vez se han validado unitariamente componentes aislados, el siguiente nivel de prueba que tiene lugar es la **validación de varios artefactos *software* integrados entre sí**, en este caso, la invocación de las operaciones de aprendizaje automático del motor desde IDbox a través del servicio API REST.

Debido a que la realización de esta prueba requiere de la invocación de los métodos del controlador de la web de IDbox, se ha creado una nueva clase `IntegrationTests` **como parte del proyecto MSTest**, el framework de Microsoft para la implementación y ejecución de pruebas en .NET. Desde esta codificación se realiza la carga de las bibliotecas de ensamblados (DLLs) de IDbox y las llamadas correspondientes para iniciar la petición de una predicción. Con la finalidad de obtener unos resultados que posteriormente puedan ser validados, se ha implementado un método auxiliar que realiza la **generación de históricos con unos valores conocidos y que no varían con el tiempo**.

A continuación se presenta la lista de pruebas aplicadas, cuyo resultado se visualiza en la figura 4.2.

- **DataIsReceived**: se realiza una petición de predicciones en base a un modelo e históricos conocidos, y se comprueba que la lista de predicciones obtenida no está vacía.

- **ExpectedValues**: en base a un modelo e históricos conocidos, se comprueba que la predicción que se obtiene como resultado es la esperada.

- **DataConsistency**: se realizan dos invocaciones de predicción con el mismo modelo e históricos conocidos. El resultado esperado es la la obtención de los mismos valores en ambas predicciones.

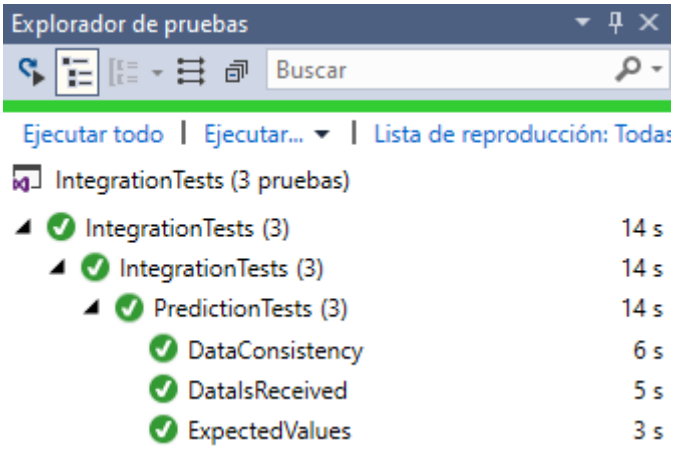

Figura 4.2: Visualización del resultado de los tests de integración a través de Visual Studio

Durante la codificación de las pruebas de integración se ha observado la presencia de dependencias inevitables con otros módulos o funcionalidades, que no aplican a estos casos de uso, y que han sido emuladas mediante *stubs*: piezas de código que **simulan el comportamiento de dichas dependencias** durante la etapa de validación.

4.3. Pruebas de aceptación

Debido a que la versión que contiene los cambios realizados aún no ha sido liberada a los clientes del producto, la **validación de usuario se ha realizado con diferentes perfiles de integrantes del equipo de IDbox**. En concreto: sesiones con el codirector del trabajo fin de grado en cada iteración del desarrollo, en las que se detectaron líneas de trabajo futuras; y pruebas específicas de cada caso de uso con una desarrolladora (cuadro 4.1) y un científico de datos (cuadro 4.2).

Caso de uso a probar	Obtener predicción.
Usuario	Desarrolladora del área de gráficas y visualización.
Descripción	El usuario solicita, a través de IDbox, la obtención de una predicción para una o más variables con datos históricos en un intervalo de tiempo personalizado.
Retroalimentación	La usuaria ha encontrado intuitivo el proceso de obtención de una predicción mediante el nuevo sistema incorporado, pues su integración se ha hecho prestando atención a conseguir la misma disposición a nivel de interfaz de usuario que cuentan los métodos de predicción ya existentes en IDbox. Como sugerencia, la usuaria propone que entre el último dato histórico y la primera predicción no se produzcan saltos en la visualización.
Acciones correctivas	En la siguiente interacción del desarrollo, se realizó la modificación en la lógica del controlador del componente del frontal para que el primer punto de predicción se visualizase como continuación del último dato histórico.

Cuadro 4.1: Prueba de aceptación del caso de uso "Obtener predicción"

Caso de uso a probar	Obtener modelo.
Usuario	Científico de datos del área de inteligencia artificial.
Descripción	Siguiendo las instrucciones del autor del motor, el usuario incorpora una nueva codificación de la lógica de aprendizaje automático y ejecuta el entrenamiento mediante la realización de una llamada a la API. Mientras tanto, se visualiza el dashboard web del motor para realizar el seguimiento de la ejecución y detectar cualquier incidencia que pueda producirse.
Retroalimentación	El resultado del entrenamiento es el deseado por el científico de datos. La jerarquía de clases y la estructura del "módulo machine learning engine provider" satisface las necesidades de ambas partes.
Acciones correctivas	No han resultado necesarias modificaciones.

Cuadro 4.2: Prueba de aceptación del caso de uso "Obtener modelo"

Capítulo 5

Explotación del sistema

Una vez que el desarrollo se encuentra validado como resultado de la correcta ejecución de las pruebas anteriormente diseñadas, el siguiente paso que tiene lugar es el **despliegue y utilización de las nuevas funcionalidades** detallado en los siguientes apartados.

5.1. Despliegue de la solución

Hasta este punto del proyecto, todo el desarrollo había sido ejecutado en el mismo equipo utilizado para las labores de desarrollo. Con la finalidad de incorporar los cambios a un entorno más cercano al de explotación, se realiza la instalación del motor de aprendizaje y la nueva versión de IDbox en un **servidor de desarrollo con características análogas a los de producción**:

1. **Instalación de Python y dependencias**: siguiendo el estudio de los distintos intérpretes de la especificación de Python mostrado al comienzo del documento, se realiza la instalación de su implementación de referencia: CPython. En este punto tiene también lugar la incorporación de todas las librerías de uso público que serán utilizadas posteriormente en el momento de ejecución.

2. **Instalación del motor**: la última codificación obtenida de este artefacto *software* es transferida al servidor de desarrollo, donde se despliega como una extensión funcional en el directorio de IDbox. Seguidamente se actualiza el fichero de configuración, adaptándolo a un configuración óptima para la máquina que lo ejecuta.

3. **Actualización de IDbox**: los cambios efectuados en la fase de integración con la plataforma de inteligencia operacional han generado una nueva versión de IDbox que debe, al igual que ocurre con el motor, desplegarse adecuadamente en el servidor de desarrollo.

4. **Instanciación del servicio de Windows**: aunque en el instante de desarrollo es suficiente con lanzar la ejecución del fichero principal que inicializa el motor, en un entorno real de explotación es necesario automatizar esta tarea. De esta forma, ante un reinicio o actualización de la máquina, el motor de aprendizaje es capaz de levantarse automáticamente, quedando así disponible para su uso sin necesidad de efectuar manualmente ninguna otra tarea adicional. Esta automatización es implementada como un servicio de Windows, que requiere de la creación de un segundo programa inicializador del motor que sea capaz de responder ante los eventos propios del servicio: parada, arranque e información de estado.

5. **Comprobación de la instalación**: una vez se han realizado todos los pasos de instalación fundamentales, no es suficiente con comprobar que los nuevos casos de uso se encuentren operativos mediante la web. En este instante tiene lugar la lectura de ficheros de log (trazas de mensajes que deja la ejecución de código) que genera tanto el motor como la web de IDbox. La finalidad de este último paso es cerciorarse de que no hay ningún mensaje de aviso o error que se esté produciendo de manera silenciosa y que en un futuro pueda causar inconvenientes visibles desde el punto de vista del usuario.

5.2. Demostración de uso

El presente apartado, a través de ejecuciones reales sobre los sistemas, muestra la explotación de la solución desde el punto de vista de los dos usuarios: el cliente y el administrador.

5.2.1. Utilización por parte del usuario final

A modo de demostración, a continuación puede observarse el resultado real de la ejecución del caso de uso "Obtener predicción" desde la web de IDbox por parte de un usuario.

Como entrada de la predicción mostrada en las figuras 5.1, 5.2, 5.3 y 5.4, se ha utilizado el modelo entrenado por defecto y la variable "CICA25", que representa la **medición de temperatura en grados centígrados de un dispositivo sensor**. El intervalo de tiempo elegido para estos históricos ha sido de un año con una resolución de los datos de una hora.

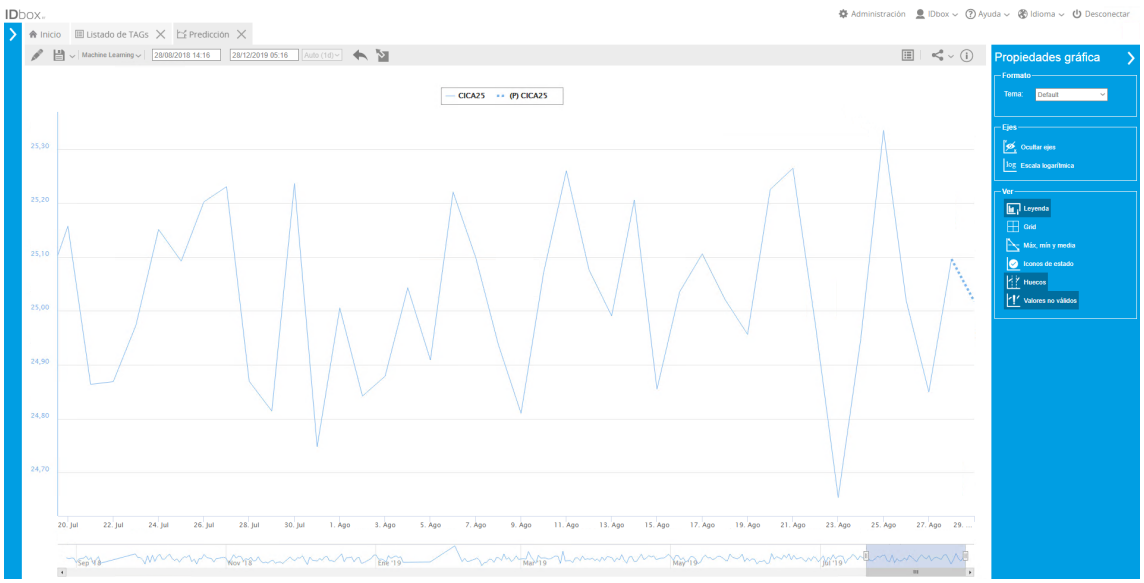

Figura 5.1: Visualización en IDbox de la solicitud de predicción para una variable

Observando con detalle el anteúltimo punto representado en la gráfica, se puede visualizar el **último dato histórico recogido**: 25,10ºC registrados el 28 de agosto de 2019 a las 00:00:00.

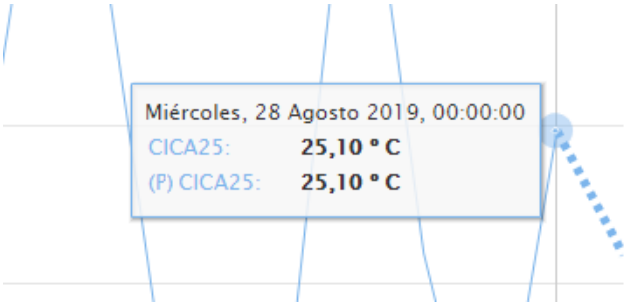

Figura 5.2: Visualización en IDbox del último punto de datos históricos

Seguidamente, y representado con una línea discontinua, se visualiza el **punto predicho para el siguiente día**: 25,02 grados centígrados, es decir, un descenso de la temperatura.

Figura 5.3: Visualización en IDbox de la primera predicción

Una vez transcurrido el 29 de agosto, se comprobó que el valor real registrado a las 00:00:00 arroja un resultado de 25,08ºC. Aunque existe una diferencia de seis décimas entre la predicción y el valor medido, se observa como **la previsión de un descenso en la temperatura** ha sido correcta.

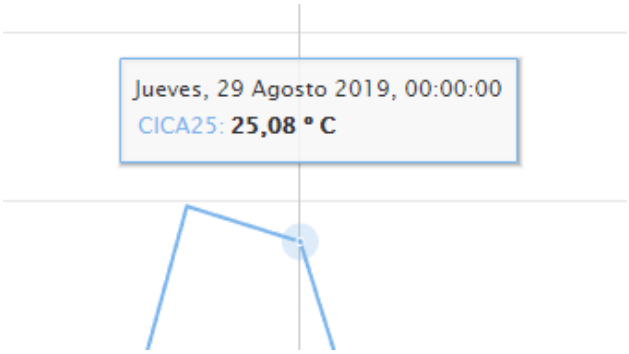

Figura 5.4: Visualización en IDbox del valor real anteriormente predicho

Como indica la especificación del caso de uso, existe la posibilidad de obtener predicciones de más de una variable. El resultado de este caso en concreto se muestra en la figura 5.5.

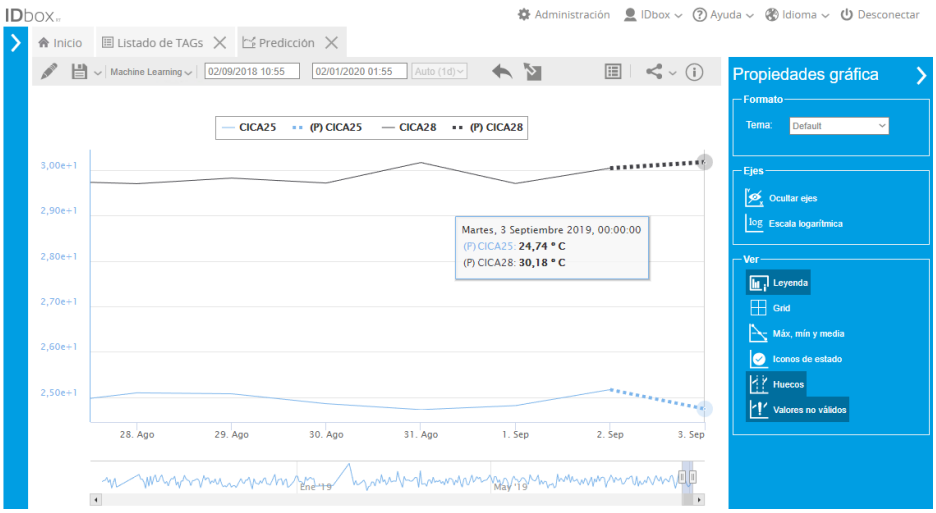

Figura 5.5: Visualización en gráfica de varias predicciones

5.2.2. Utilización por parte del personal de administración

En paralelo a estas operaciones de aprendizaje automático, el equipo de IDbox cuenta con acceso a la API y al *dashboard web* para la **monitorización, administración básica y diagnóstico de problemas del motor**.

La figura 5.6 muestra el **estado del motor,** a través del tablero web, durante la ejecución de varias predicciones solicitadas desde la web de IDbox. Se observa cómo se encuentran ocupados en su totalidad los distintos *workers*, la fase de ejecución de las peticiones: en este caso completadas y asignadas, así como los mensajes de la traza de ejecución.

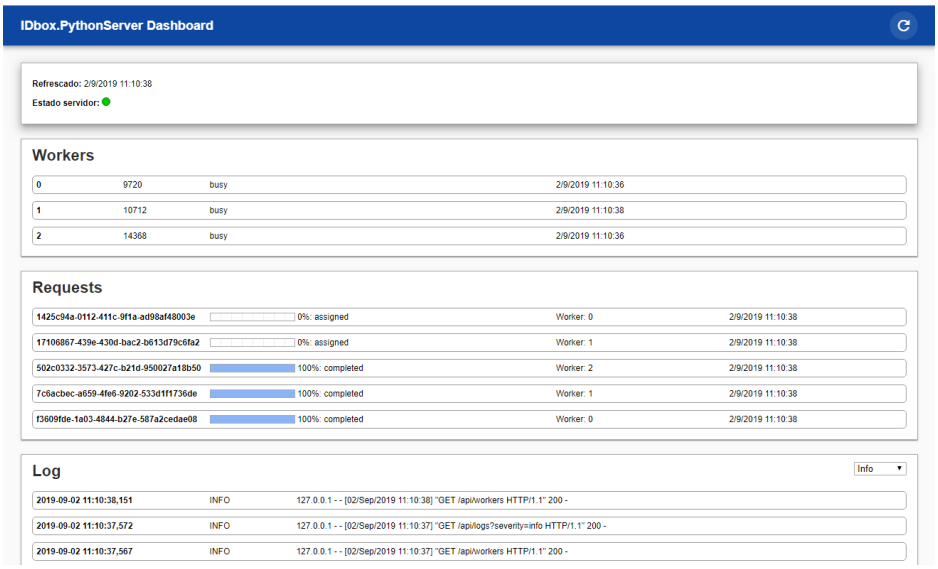

Figura 5.6: Monitorización del estado del motor a través del *dashboard web*

En caso de una situación anómala, **el uso de esta herramienta resulta de un gran interés para conocer el origen del problema**. Como ejemplo ilustrativo, la figura 5.7 muestra el estado del motor tras el caso supuesto del reporte de una incidencia con las predicciones por parte de un cliente. Se observa que no existe el *worker* 1, y que hay una tarea cancelada.

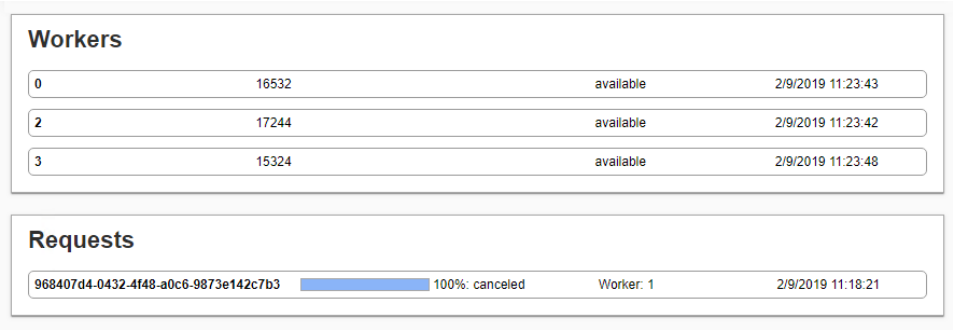

Figura 5.7: Visualización de una situación anómala a través del *dashboard web*

Comprobando el **historial de ejecución**, que muestra la figura 5.8, se observa qué ha ocurrido: la instancia de trabajo con id 1 ha finalizado de manera forzosa debido a un error, dejando la tarea cancelada. Como reemplazo, el supervisor de estado del sistema ha levantado un nuevo *worker* con identificador 3.

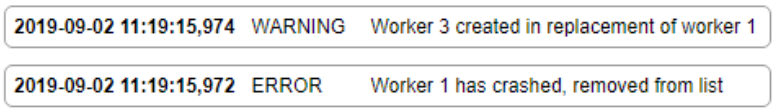

Figura 5.8: Lectura de la traza de ejecución desde el *dashboard web*

Por último, la figura 5.9 muestra la **utilización directa de los recursos de la API**, en concreto, la visualización de los *workers* mediante las herramientas de desarrollo de Chrome. Este modo de uso puede ser de interés para aquellos miembros del equipo con perfil técnico que deseen ejecutar acciones avanzadas u obtener información concreta directamente en formato JSON.

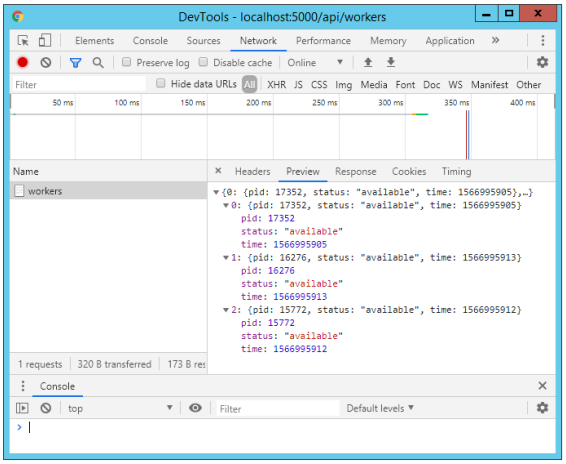

Figura 5.9: Visualización de un mensaje JSON obtenido a través de la API

Capítulo 6

Conclusiones y trabajos futuros

Como capítulo final de la presente memoria, se ofrece una reflexión sobre el trabajo mostrado en las secciones anteriores, así como un análisis de las futuras líneas de mejora que podrían aplicarse al proyecto a partir del estado presentado.

6.1. Conclusiones

El presente proyecto fin de grado tiene como objetivo la invocación de código Python específico desde C#, cumpliendo con unos requisitos de calidad en cuanto al control de errores, de estado y el uso de grandes cantidades de datos. Para ello **se han empleado hasta tres lenguajes de programación** de diferente naturaleza: Python, C# y Javascript. A estos se suman también los de marcado con finalidad de diseño: HTML y CSS. Además, se han abordado **una gran variedad de destrezas y áreas de conocimiento de ingeniería informática** como multiprocesamiento mediante procesos, threads y su comunicación mediante colas de memoria; diseño de una interfaz de comunicación e intercambio de mensajes estructurados entre sistemas dispares; conceptos de inteligencia artificial y cálculos; y la captura de requisitos para la posterior construcción. Esta **meta se ha alcanzado completamente** con dos fases diferenciadas: una primera de análisis, investigación y prototipado; y una segunda de diseño, implementación e integración.

Personalmente, la realización de este proyecto ha sido **una experiencia realmente enriquecedora**. Completar todas sus partes me ha permitido aprender sobre un dominio hasta ahora desconocido para mí, el campo del **aprendizaje automático en la inteligencia artificial**. En cuanto a desarrollo, al ser empleado Python como plataforma para implementar las exigentes características del núcleo del proyecto, he terminado alcanzando buena destreza en su manejo tras no haberlo utilizado con anterioridad. De igual modo, he tenido la oportunidad de **consolidar el aprendizaje en C#** comenzado durante el grado y **dar uso a los conocimientos sobre desarrollo web**.

Aunque no es el primer proyecto *software* que desarrollo fuera del ámbito académico o profesional, sí es el primero realizado en una empresa desde una fase de análisis a bajo nivel hasta su completo diseño e integración, **tomando todas las decisiones relativas al uso de tecnologías, herramientas y diseño de procesos**. Como resultado, la solución presentada a lo largo del documento no se trata de un estudio teórico, prototipo, o prueba de concepto, sino **una implementación real cuya integración ya se encuentra proporcionando retroalimentación positiva en la empresa**.

6.2. Trabajos futuros

Aunque los casos de uso expuestos presentan un funcionamiento correcto, se han identificado unas líneas de trabajo futuro para **mejorar los procesos y ampliar el alcance**.

- Actualmente, las predicciones se basan siempre en un modelo de regresión entrenado durante el trabajo fin de grado. De cara a una puesta en producción, sería interesante la posibilidad de **asignar diferentes modelos entrenados** según la cantidad de datos históricos, tipo de variable u otras situaciones. Esto podría realizarse en algunos casos automáticamente, o elegido de manera manual por el cliente.

- Otra línea identificada es la **mejora del proceso de entrenamiento del modelo**, en el que actualmente es necesario hacer manualmente uso de la API para invocar la operación. Esta tarea podría facilitarse mediante la creación de una herramienta específica que proporcione una abstracción a este acceso programático ya existente.

- Aunque tanto el motor como la integración con IDbox han sido desarrollados para soportar el manejo de varios puntos de predicción, actualmente solo se genera el punto siguiente. Se propone la **mejora de la lógica de inteligencia artificial para que se proporcionen varios resultados**.

- Por último y de cara a una puesta en producción, a nivel de la infraestructura del sistema se propone la **sustitución del servidor integrado de Flask por su integración con IIS**, el servidor de referencia en entornos Windows.

Bibliografía

[1] Bureau International des Poids et Mesures. *The International System of Units (SI).* [Consulta: 26-07-2019]. Disponible en: https://www.bipm.org/utils/common/pdf/si-brochure/SI-Brochure-9-EN.pdf.

[2] IDbox RT. *Sectores - IDbox RT.* [Consulta: 29-07-2019]. Disponible en: https://idboxrt.com/sectores/.

[3] IDbox RT. *Producto - IDbox RT.* [Consulta: 29-07-2019]. Disponible en: https://idboxrt.com/producto/.

[4] IDbox RT. *Qué es la Inteligencia Operativa.* [Consulta: 29-07-2019]. Disponible en: https://idboxrt.com/inteligencia-operativa/.

[5] Python. *The Python Standard Library — Python 3.7.3 documentation.* [Consulta: 03-07-2019]. Disponible en: https://docs.python.org/3/library/index.html.

[6] Python. *PythonImplementations - Python Wiki.* [Consulta: 03-07-2019]. Disponible en: https://wiki.python.org/moin/PythonImplementations.

[7] Python. *Welcome to Python.org.* [Consulta: 02-09-2019]. Disponible en: https://www.python.org/.

[8] Stackless Python. *Home · stackless-dev/stackless Wiki · GitHub.* [Consulta: 02-09-2019]. Disponible en: https://github.com/stackless-dev/stackless/wiki.

[9] PyPy. *PyPy - What is PyPy?* [Consulta: 02-09-2019]. Disponible en: https://pypy.org/features.html.

[10] MicroPython. *The MicroPython language — MicroPython 1.11 documentation.* [Consulta: 02-09-2019]. Disponible en: http://docs.micropython.org/en/latest/reference/index.html.

[11] CircuitPython. *What is CircuitPython? — Welcome to CircuitPython! — Adafruit Learning System.* [Consulta: 02-09-2019]. Disponible en: https://learn.adafruit.com/welcome-to-circuitpython/what-is-circuitpython.

[12] IronPython. *IronPython.net / .* [Consulta: 02-09-2019]. Disponible en: https://ironpython.net/.

[13] Python for .NET. *Python for .NET — pythonnet.github.io.* [Consulta: 02-09-2019]. Disponible en: http://pythonnet.github.io/.

[14] Jython. *JythonFaq/GeneralInfo - JythonWiki.* [Consulta: 02-09-2019]. Disponible en: https://wiki.python.org/jython/JythonFaq/GeneralInfo.

[15] Martin Fowler. *EvansClassification.* [Consulta: 23-08-2019]. Disponible en: https://martinfowler.com/bliki/EvansClassification.html.

[16] Python. *queue — A synchronized queue class — Python 3.7.3 documentation.* [Consulta: 08-07-2019]. Disponible en: https://docs.python.org/3.7/library/queue.html.

[17] Python. *multiprocessing — Process-based parallelism — Python 3.7.4 documentation.* [Consulta: 08-07-2019]. Disponible en: https://docs.python.org/3.7/library/multiprocessing.html.